Stereo Vision and Mapping with Unsynchronized Cameras

by

Ray C. He

S.B., Electrical Engineering and Computer Science
Massachusetts Institute of Technology (2007)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2008

Author .....
Department of Electrical Engineering and Computer Science
May 23, 2008

Certified by .....
Jonathan How
Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Professor of Electrical Engineering
Chairman, Department Committee on Graduate Students

Stereo Vision and Mapping with Unsynchronized Cameras

by

Ray C. He

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2008, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Environmental awareness is an important prerequisite for autonomous behavior in vehicles. Without it, robots are unable to react to unknown surroundings and require extensive human input for tasks such as target identification and obstacle avoidance. This would negate many of the advantages of having an autonomous system. Giving a vehicle the ability to map its surroundings and use the data effectively will allow humans to spend less time scanning the vehicle's video feed and providing direct navigational commands. This thesis details the development of a real-time, extensible vision and mapping system that provides an interface for control systems to access details of the map. It addresses the problems of image capture, signal noise, and three-dimensional map storage. It extends existing real-time stereo mapping systems by tolerating unsynchronized stereo cameras. Results indicate that synchronization allows the system to locate points significantly more accurately than the system without synchronization. When compared with a monocular mapping system, synchronized stereo provides a more detailed map and will tolerate more erroneous localization data. Because it is developed with an abstract localization system, this system is designed to be modular and easily extensible.

Thesis Supervisor: Jonathan How
Title: Professor of Aeronautics and Astronautics

Acknowledgments

The author would like to thank the people without whom this thesis could not have existed. His family, a limitless source of encouragement for the past quarter-century; his friends, a diverse group of passionate individuals; and the Aerospace Controls Lab, a community of researchers possessing uncommon skill and contagious motivation. Specifically, the author would like to thank his advisor, Professor Jonathan How, for his guidance over the past year and his assistance writing this thesis. He is grateful for Brett Bethke and Spencer Ahrens, extraordinary colleagues who have provided assistance in developing the ideas within this thesis.

This thesis was made possible by a research grant from the Boeing Company, whose generous support demonstrates a commitment to innovation that will lead to new technologies that improve the human experience.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Autonomous robots have increased in usage and applicability in the past decade. The delegation of tasks to an autonomous system and simultaneous decrease in human supervision is made possible by increased sophistication in the systems that control the robots. With the advent of faster processors, more precise sensors, better actuators, and more robust controllers, these robots are now capable of performing complex tasks. This substitution is motivated by the nature of certain missions, particularly ones that can best be described as mundane, repetitive, or dangerous. Examples of these tasks range from chore automation, reconnaissance, Explosive Ordinance Disposal (EOD), combat support, to firefighting.

Modern day development and deployment of household robots include the iRobot Roomba [9], a vacuum cleaning robot[1], and the Friendly Robotics' Robomow [36], a lawnmowing robot. The level of automation in these robots only requires commands to begin or a set schedule to complete their tasks, though the Robomow does require a wire "fence" to mark the boundaries of the target environment. Military research and development of support robots has resulted in a number of applications, such as reconnaissance robots. These include the automation of Unmanned Aerial Vehicles (UAVs), exemplified by the tracking of ground vehicles [3], and the automation of vehicles designed to traverse a complex environment, the objective of vehicles deployed in the DARPA Urban Challenge [7].

[1]Other model lines created by iRobot allow for tasks such as floor washing and pool cleaning.

(a) iRobot's Roomba, an autonomous vacuum-cleaner

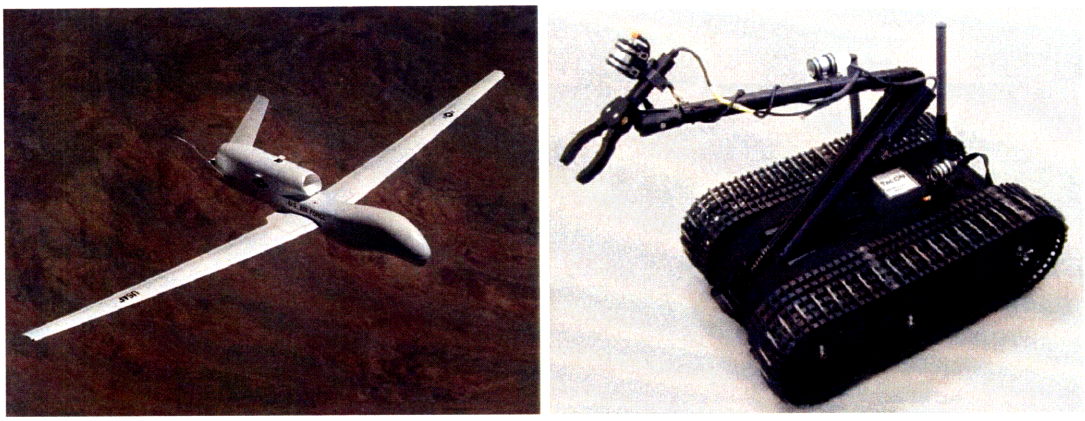
(b) Friendly Robotics' Robomow, an autonomous lawnmower

Figure 1-1: These robots run autonomously and only require high-level human supervision, such as defining a lawn boundary and scheduling operation, to complete their missions.

In addition to these automated applications, remote-controlled drones are increasingly used for EOD, fire support, and casualty recovery [44], such as Foster-Miller's bomb disposal and "special weapons observation remote reconnaissance direct action system" robots [11]. While not currently automated, these systems lend themselves to automation as the military moves to greater automation pushes for Unmanned Ground Vehicles (UGVs). UAVs and UGVs are also good candidates for further automation as current operational models, such as Northrop Grumman's Global Hawk [27] and General Atomics' Predator [12] and MDARS Humvees [13], require a human operator to scan for targets.

1.1 Motivation

Because the current EOD, fire support, casualty recovery, and UAV/UGV systems still require significant human intervention for many of their capabilities, these tools have not reached their full potential usefulness. Development and integration of in-



(a) Northrup Grumman's Global Hawk UAV, designed for surveillance.

(b) Foster Miller's Talon, configured for remote EOD.

Figure 1-2: While these unmanned robots are currently remotely controlled, the nature of their missions make them candidates for automation.

telligent control systems is necessary to exploit their potential to relieve a human operator of tedious tasks, such as target identification and navigation. These control systems will require accurate estimates of the state of the world around the robot in order to react to changes or to perform a task with undetermined obstacles. Environmental awareness is thus a crucial component to realizing robust controllers that are capable of operation in a poorly-constrained world.

This thesis addresses the development of an environment mapping system that uses images gathered from two cameras mounted in a stereo configuration and localization data to populate a map of the cameras' surroundings. It utilizes machine vision algorithms to find correspondences between each temporal frame and spatially between the two cameras. A synchronization technique is used to estimate feature movement intra-frame to approximate stereo correspondence for two images in the presence of asynchronous cameras, a common problem when pairing together Commercial, Off-The-Shelf (COTS) cameras not designed for stereo operation. The system abstracts the localization system and provides an API for controllers that need access to a map, allowing various implementations to be modularly used.

1.2 Background

A significant amount of research in Simultaneous Localization and Mapping (SLAM), machine vision, and digital image processing preceded this thesis, which builds on developments in the various fields.

Digital image processing uses computational techniques to improve images for the purposes of storage efficiency, human viewing, computer vision [22]. For computer vision, the techniques include filtering for noise to provide a more consistent representation of an image and selectively modifying contrast in certain regions of the image to aid in global edge or feature detection. Techniques of digital image processing can be extended to object detection and recognition, which can use a variety of transforms to decompose an image to examine it for identifying criteria [14, 23, 24]. One commonly used object detection technique makes uses of the Haar transform [21], and has been implemented for detection through the use of series of classifiers [19].

In the realm of mapping and localization, a variety of feature extraction and grouping methods have been used. While many approaches have used precise location of environment objects through the use of laser rangefinders [10, 15, 16, 25, 26], several have employed vision-based methods both involving single-camera and stereo configurations [20, 37]. The advantages of the vision approach include weight and cost, with a set of commercial stereo cameras costing under 1000 USD[42][2] compared to a typical rangefinder such as the SICK LMS 200 weighing in at over 5000 USD and 4.5 kg[39]. The basic premise of mapping using vision is to determine a set of landmarks and determine their location using some combination of parallax and odometry and multiple measurements. From these measurements, the SLAM problem is solved but iteratively updating the location of the robot using optical flow or odometry, observing landmarks and finding a difference between the new observation and the old observation plus the movement update, and updating an estimate of robot location based on this difference [25]. Finding landmarks usually involves extracting features from sensor input, processing them into landmarks, and storing the landmarks in

[2]The set of cameras used for this thesis retail at 250 USD per camera and have a operating weight of 180 grams when stripped of unrelated components.

data structures such as occupancy grids, topographical maps, or as lists or binary trees [26, 32].

Ideally, the features that form landmarks can be detected and oriented using a scale-invariant feature detector [23], though most implementations of such feature detectors cannot operate in real time without the use of specialized hardware for performing floating point vector operations, such as graphics processing units [40]. Other ways of locating features include Shi and Tomasi's Good Features to Track [38] and the Harris detector [17], both of which have significantly faster running times but lack the scale invariance of [23]. After feature detection, correspondences in the matching stereo image can be accomplished using the Lucas-Kanade optical flow algorithm, implemented in OpenCV [4, 19]. The optical flow algorithm can also be used to find correspondences between adjacent frames in time from a single camera, which is representative of the camera's movement.

With the previously mentioned stereo vision approaches using dedicated commercial stereo cameras or two wired video cameras mounted in stereo, no method for using asynchronous, high-latency cameras currently exists to the best of our knowledge. Because of communication delays with wireless cameras using digital transmission, tolerating asynchronous stereo images is necessary if discrete wireless cameras are to be used.

1.3 Objectives

The objectives of this thesis are to develop a real-time, modular computer vision system that uses unsynchronized cameras for the purpose of environment mapping by providing an feature map of an indoor environment. While the goal is not to implement a full SLAM solution, the success of this system will be determined by the ability of the system to build a feature map given localization information along with asynchronous stereo camera data, and the ability to highlight predetermined objects on the map as *objects of interest.*

1.4 Development Framework

The Aerospace Controls Laboratory (ACL) has worked for the past few years on a test-bed for developing control systems and health management systems for groups of autonomous vehicles. Dubbed the Real-time indoor Autonomous Vehicle test ENvironment (RAVEN), the system uses the VICON motion capture system [41] to gather precise spatial location data, which it passes on to individual vehicle control systems. Currently, the autonomous vehicles used include a variety of both ground and air vehicles, including RC trucks, quadrotor helicopters, as well as rovers and planes. Major areas of research have included health management and control systems for a variety of flying vehicles. The algorithms and procedures created in this test-bed can be exported to applications including military reconnaissance, ad-hoc wireless networking, and automated mapping. This thesis uses ACL's VICON system for absolute localization to test the synchronized stereo mapping modularly. Additionally, it is built modularly alongside a vision project designed to localize a robotic vehicle.

1.5 Outline

The structure of this thesis follows the system diagram in Figure 1-3. The implementation and performance of the image acquisition, pre-processing system, and stereo synchronization system is described in Chapter 2. Chapter 3 presents the object detection module for the mapper and a classifier creation frontend designed to work with the system. Chapter 4 describes the implementation and performance of the mapping system. Finally, Chapter 5 summarizes the project's strengths and shortcomings and outlining a path for future development. Appendix A details the software implementation for each part of the system and Appendix B describes an adaptive noise reduction technique that was created for use with analog transmission cameras. While the cameras that were eventually selected used digital transmission, the adaptive filter would be useful as a noise reduction module for this system if the cameras were replaced.

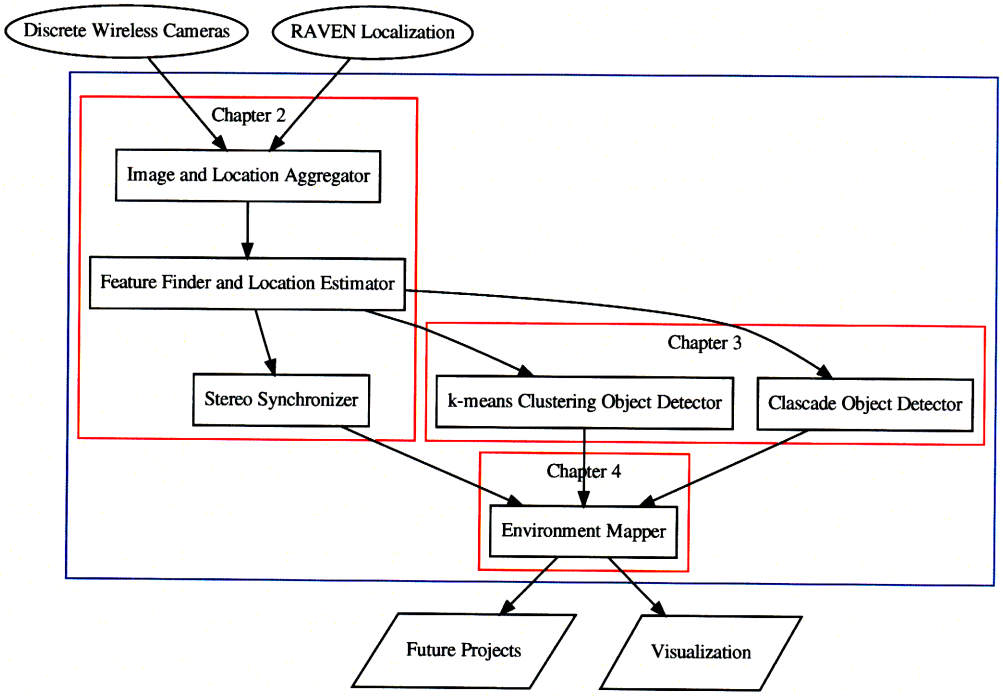


Figure 1-3: System overview of image processing, object detection, and mapping.

Chapter 2

Image Acquisition and Processing

This chapter describes a real-time, parallel image acquisition and processing system designed to process the input from stereo cameras. It is the largest component of this thesis and fits into the complete system as seen in Figure 2-1. While tested and used on two cameras, this application features support for multiple cameras, and is only limited by network bandwidth and memory for temporary images. Because the system is designed for use with moving stereo cameras, it maintains an adjustable buffer of optical flow information for each vision stream. These buffers are for use by the camera synchronization and stereo correspondence modules in the processing pipeline.

2.1 Image Capture System

Camera selection presents a real problem for a stereo environment mapping application using COTS cameras. At the time of this thesis, all commercially available stereo cameras, such as the Point Grey Research Bumblebee Series [33] and Videre Design's product line [42], shown in Figure 2-2, utilize the IEEE-1394a interface to communicate with computers. No wireless solution existed for IEEE-1394, so the decision was made to use two individual wireless cameras mounted in a stereo configuration. The options included a variety of video cameras using analog transmission, including the Draganflyer 2.4GHz Micro Wireless Video Camera system, shown in

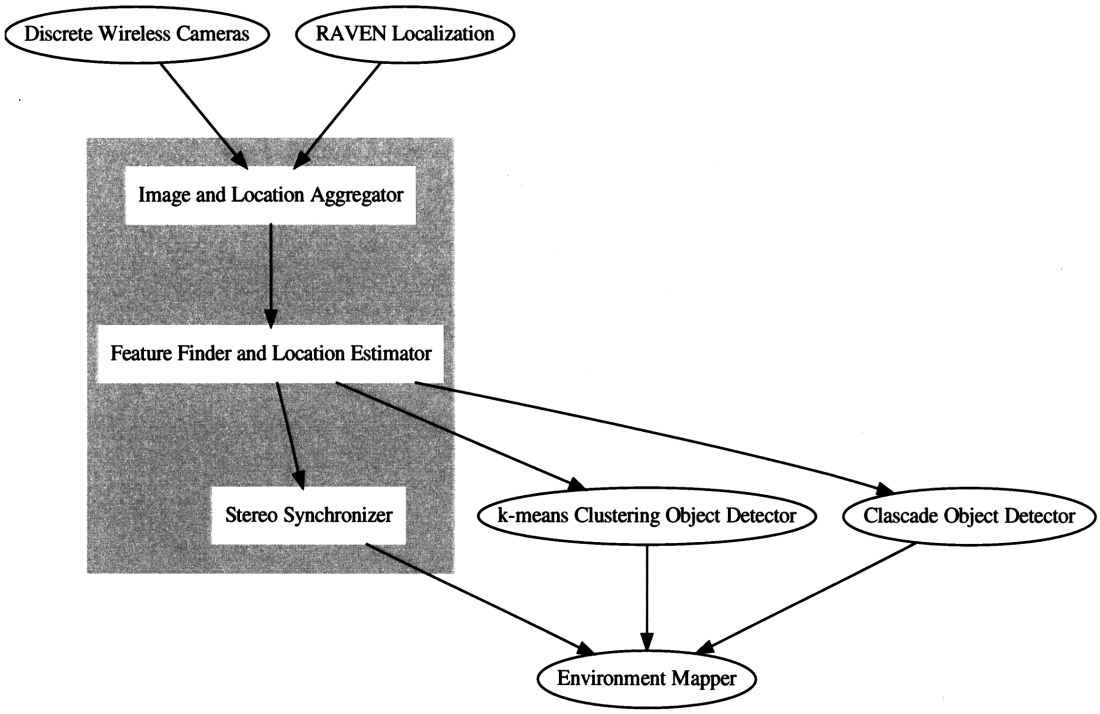


Figure 2-1: System components of the image acquisition and processing system.

Figure 2-3 [8]. While these cameras provided adequate video outside, indoor wireless interference caused noise that unacceptably reduced the accuracy of object detection and stereo correspondence. An adaptive interpolation and median filter, detailed in Appendix B, was developed to compensate for the noise. Unfortunately, these techniques could not improve the video streams to a point where stereo correspondence worked at an acceptable level.

2.1.1 Camera Configuration

The camera eventually selected for this thesis is the Panasonic BL-C131 Wireless Network Camera, which allows communication over 802.11g and provide video that can adequately used for stereo correspondence. The cameras were stripped of their camera servos to reduce weight and mounted in stereo configuration, as seen in Figure 2-4. The cameras have a 49° horizontal field of view and a 37° vertical field of view, a

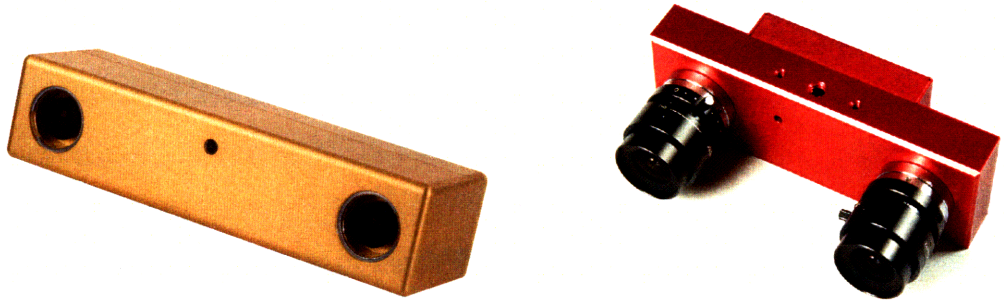


Figure 2-2: Point Grey Research's Bumblebee (left) and Videre Design's stereo cameras only use the IEEE-1394 interface and are incapable of lightweight wireless operation.

Figure 2-3: Draganfly Innovations Inc's 2.4 GHz analog transmission cameras are subject to wireless interference that results in image streams unsuitable for stereo correspondence calculation indoors.

lens fixed at infinity focus, and support resolutions up to 640x480 at frame rates up to 30 fps with good lighting [31]. The baseline for the cameras was set as 11cm, a width chosen based on an average of commercially available stereo camera configurations. The advantages of a closer baseline are less image disparity and the ability to measure closer depths, while a wider baseline results in better measurements of disparities farther away. Unfortunately, these cameras do compromise on weight, at 190 g[1] compared to the 20 g analog cameras.

[1] After stripping out unnecessary components and leaving a protective casing.

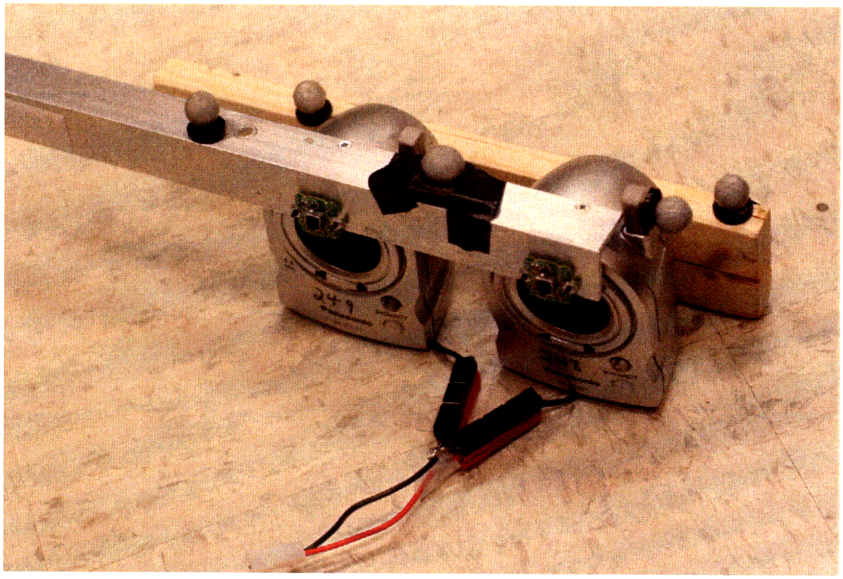


Figure 2-4: Two Panasonic BL-C131A cameras stripped of their servos and mounted in stereo configuration.

2.1.2 Raw Image Capture

The BL-C131A's have onboard webservers that allow images to be streamed from them via hypertext transfer protocol (HTTP) requests in a variety of formats and framerates. The image configuration and general camera configuration are both handled via variables passed using `HTTP_GET`. A image fetcher application was written to request images from the camera as a `multipart/x-mixed-replace` jpeg stream at 320x240 resolution. This request type works with a server-push model and keeps an open connection to the camera through which images are streamed, separated by a delimiter. The advantage of using the server-push model is theoretically higher throughput since the image receiver does not have to make a new HTTP request for every image [1]. A test of this advantage can be seen in Table 2.1, where the streaming request got higher frame rates in both low and high lighting conditions. The difference is not as great in low light because the frame rate is limited by the speed of the electronic shutter, which must be kept open longer in low light.

Table 2.1: The results of testing image capture using both methods of image capture. For the purposes of the test, both implementations were run on a dedicated 100 mbit local area network with latencies around 0.8 ms

Typical Framerates with Different Methods	
Indoor Light, `multipart/x-mixed-replace`	15.2
Indoor Light, image pull	12.3
High Light, `multipart/x-mixed-replace`	29.4
High Light, image pull	13.2

2.2 Image Processing System

The images are processed upon retrieval according to the image pipeline detailed in Figure 2-5, where the blue box surrounds the main components of the image processing system. In separate threads, each camera is continuously queried for images and preprocessed for sparse optical flow, which is used to synchronize the cameras. At the time of reception, each image is converted to black and white, histogram-equalized for increased contrast, and stamped with the time the image was received. The left vision stream maintains a set of tracking features through the use of OpenCV's Harris detector implementation, an alternate option of `cvGoodFeaturesToTrack`. The typical mode of operation is to find new features only in areas of the image which did not inherit features from the previous frame, accomplished through the use of `cvGoodFeaturesToTrack` with a mask over the inherited features. The inherited features are found by using sparse optical flow of features from the previous image. This is accomplished by the use of OpenCV's `cvCalcOpticalFlowPyrLK`, an implementation of the Lucas-Kanade optical flow calculator. This optical flow algorithm is also used to find correspondences between left and right frames when calculating feature locations.

2.2.1 Uniquely Determining Features

For later use by continuous location estimation outside of a single frame, each feature must have correspondence in time, between frames of a sequence for a single camera,

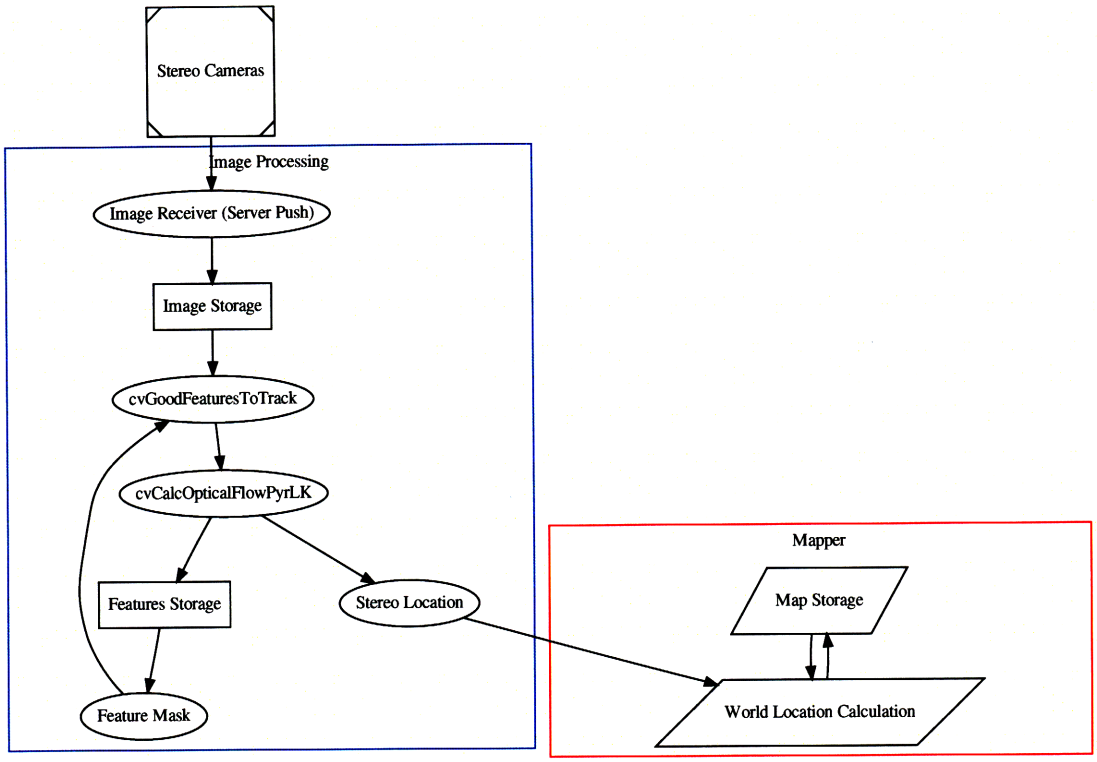


Figure 2-5: The image processing system is boxed in blue. Note that elliptical components represent methods and operations while rectangles represent data structures and storage.

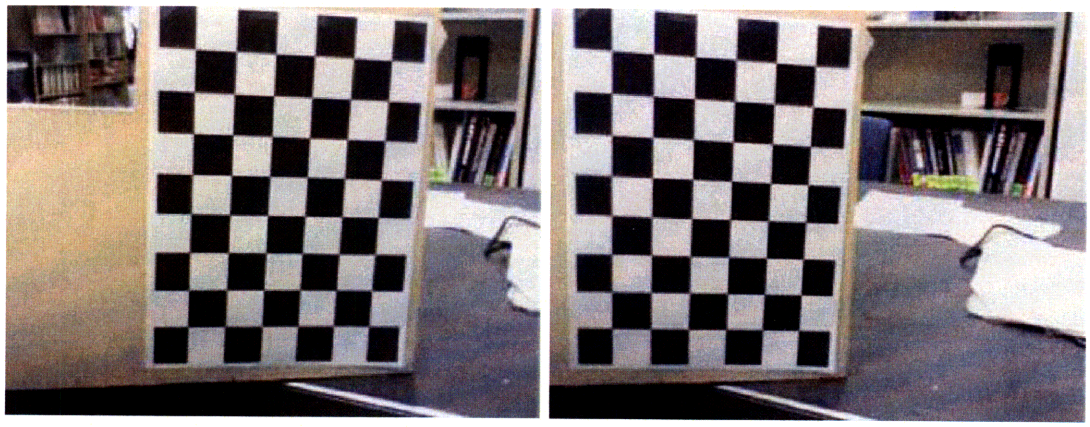
as well as space, between corresponding frames of the left and right camera. This is accomplished through the use of the aforementioned optical flow algorithm, with the output features inheriting the unique identifiers of their parents. These unique IDs are determined initially using a hash of the timestamp and a number of the feature when it was first detected through the use of the corner detector. Because a camera image without location or mapping data cannot re-recognize features that leave a frame without using something similar to the computationally expensive scale-invariant algorithm mentioned in [23], features that come back into view are tagged with a unique identifier until add feature calls to the mapper module, described in Chapter 4, determines that it is a previously seen feature.

2.3 Camera Synchronization

The cameras are synchronized using recalculated optical flow data from a window of the last N frames from each camera. Within this window, the left camera's $\lfloor N/2 \rfloor$-th frame is used as a keyframe, which is the frame of interest for determining stereo depth and doing environment mapping. Optical flow is calculated from this keyframe for the $N-1$ other frames for the left cameras and for the N frames from the right camera. The use of the keyframe is designed to ensure that the flow calculated between each frame comes from the same set of sparse features. When a feature has left the field of view for one of the surrounding frames, `cvCalcOpticalFlowPyrLK` reports its feature status as `False`. Features that are missing from any of the frames are left out of synchronization results. Consequently, features that may have appeared because of random noise or occlusions during movement are conveniently filtered out. This also imposes a limit on the window size N, as too great of an N will keep features from being detected during large camera movements. For this project, N was chosen to be 5 to reduce the amount of processing while providing a reasonable window for temporal delay in receiving frames. This will compensate for a frame lag of up to three frames using the method described below, which roughly corresponds to .2 seconds at a framerate of 15 Hz. This is reasonable for this project as network latencies for the wireless cameras are in the 90 ms range on average. Selecting a higher N would trade asynchronous camera tolerance for greater processing time, as optical flow is calculated once for every N.

The result of the optical flow calculation is a set of corresponding features that are present in all left and right frames. For each feature, this creates a set of optical flow vectors over time in the corresponding cameras. This optical flow can be synchronized in a number of ways, including using the timestamps associated with each optical flow or by using the flow vector shapes to determine a movement. This process is made simpler after a calibration of the cameras to constrain the transform, which is accomplished through the use of the Camera Calibration Toolbox for MATLAB [5]. Only after correcting for the rotation and vertical translation of the right camera, can

t]



(a) Left Camera Image (b) Right Camera Image

Figure 2-6: Examples of a corresponding set of left and right images uses for calibration.

a meaningful estimation of the real feature disparities take place.

2.3.1 Camera Calibration Results

Running the camera calibration toolbox on the stereo cameras used in this thesis resulted in the following output. The calibration was done using a printed checkerboard downloaded from [5]. The printed checker squares were confirmed to be 30 mm by 30 mm in size and images were taken with both cameras for the same checkerboard orientation, as seen in Figure 2-6. Ten corresponding images were used for the calibration, which resulted in output from the toolbox seen in Figure 2-7 and the camera orientation in Figure 2-8, which includes orientations of the checkerboards relative to the cameras.

The relevant output from the camera calibration is the rotation and translation vector

$$T = [-103.3, -0.9, -2] \pm [0.2, 0.2, 1.4] \tag{2.1}$$

in units of millimeters. R is calculated from om using the Rodrigues rotation formula

```
Intrinsic parameters of left camera:
Focal Length:          fc_left = [ 347.02702   350.52140 ] +/- [ 2.23741   2.36609 ]
Principal point:       cc_left = [ 145.14075   112.04699 ] +/- [ 2.99743   2.71451 ]
Skew:             alpha_c_left = [ 0.00000 ] +/- [ 0.00000  ]   => angle of pixel axes = 90.00000 +/- 0.00000 degrees
Distortion:            kc_left = [ 0.05153   -0.17787   0.00311   -0.00285  0.00000 ] +/- [ 0.02331   0.07212   0.00243   0.00327  0.00000 ]

Intrinsic parameters of right camera:
Focal Length:         fc_right = [ 345.12611   348.78894 ] +/- [ 2.23489   2.33090 ]
Principal point:      cc_right = [ 157.78080   113.68213 ] +/- [ 2.93064   3.00314 ]
Skew:            alpha_c_right = [ 0.00000 ] +/- [ 0.00000  ]   => angle of pixel axes = 90.00000 +/- 0.00000
degrees Distortion:   kc_right = [ 0.06959   -0.26946   0.00042    0.00211  0.00000 ] +/- [ 0.02946   0.14443   0.00284 0.00327  0.00000 ]

Extrinsic parameters (position of right camera wrt left camera):

Rotation vector:             om = [ 0.00215   -0.01734  0.01785 ] +/- [ 0.00918   0.01008  0.00072 ]
Translation vector:           T = [ -103.31057   -0.86215  -2.38490 ] +/- [ 0.22995   0.19640  1.38399 ]
```

Figure 2-7: Output from the camera calibration toolbox for MATLAB.

to be:

$$R = \begin{bmatrix} 0.9997 & -0.0179 & -0.0173 \\ 0.0178 & 0.9998 & -0.0023 \\ 0.0174 & 0.0020 & 0.9998 \end{bmatrix}$$

This rotation matrix means that the right camera needs to by rotated by R and translated by T to get in the frame of the left camera: $C_L = RC_R + T$, where C_L and C_R are the left and right camera locations. This translation corresponds to the manual measure of the camera baseline to be 10 cm versus 103.3 mm in x, and aligned versus 0.9 mm and 2 mm in y and z.

2.3.2 Synchronization by Time

Because each captured frame is tagged by a receive time, motion interpolation can establish an estimate for where a feature is given where it was in the frames surrounding it. Second-order temporal interpolation is used over the right camera's frames to determine the location of the keyframe's direct temporal counterpart. For the case of $N = 5$ frames,

$$\tilde{f}_r = \frac{\sum_{i=0}^{N} |t_r(i) - t_l| f_r(i)}{\sum_{i=0}^{N} |t_r(i) - t_l|} \tag{2.2}$$

$\tilde{f}_r$ is the right frame's feature location estimate, t_l is the timestamp of the left keyframe, $f_r(i)$ is the right frame's feature location in the i-th frame at time $t_r(i)$.

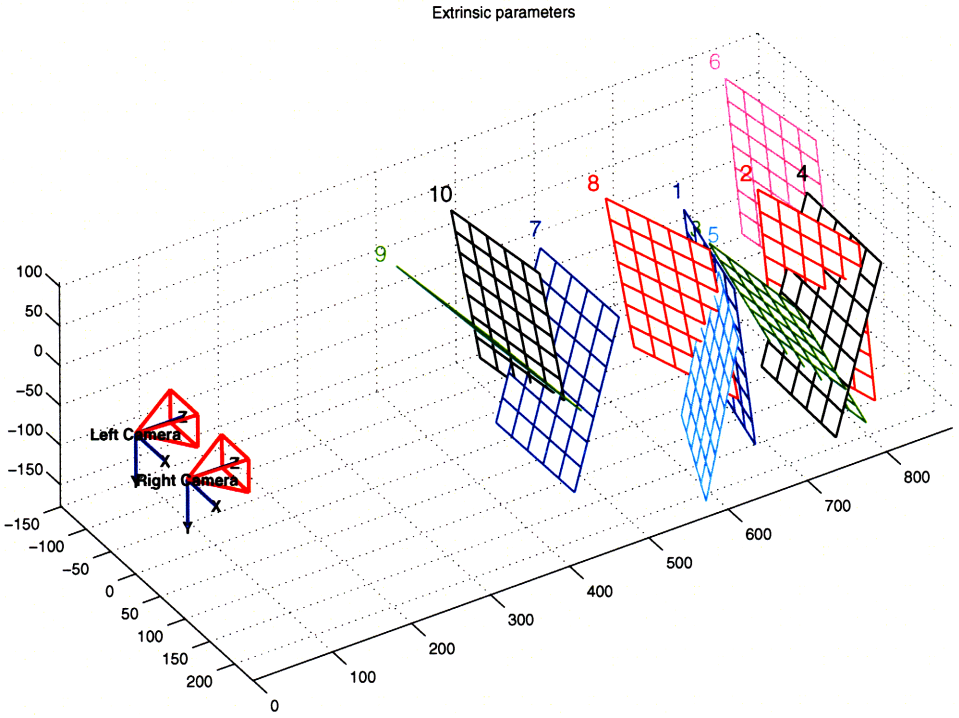


Figure 2-8: Image showing the orientation of the stereo cameras, determined by the stereo calibration tool.

2.3.3 Synchronization by Minimizing Curve Error

Assuming the pinhole camera model and a small range of motion over $N = 5$ frames, the flow vectors will form a curve and the right optical flow curve will be an approximately a translation of the left optical flow curve. Each corresponding feature in time is an asynchronously sampled point on this curve. To detect the real disparity for each feature between the two images, that affine transform needs to be estimated. Since the features from the right camera have been transformed to correct for rotation, only linear displacements need to be estimated. To do this, each left feature's corresponding optical flow curve is fit with a third order polynomial using Python's numeric python module's polynomial fitting tool, `numpy.polyfit`. This yields an equation in the form:

$$f(x) = ax^3 + bx^2 + cx + d \tag{2.3}$$

Here x, $f(x)$ are the x,y-locations of the pixel and a, b, c, and d are the constants. Next, the x_o, y_o combination that minimizes the squared error, $\mathbf{E}$, is found using Newton's method.

$$\mathbf{E} = \sum_{i \in \mathcal{R}} (y_i - f(x_i - x_o) - y_o)^2 \tag{2.4}$$

$$v_0 = [x_o, y_o]^T \tag{2.5}$$

$$v_{n+1} = v_n - \frac{\nabla \mathcal{E}}{\mathcal{H}\mathbf{E}} \tag{2.6}$$

v_n is initialized to v_0 in Eq. 2.5 and then iterated until a convergence threshold via Eq. 2.6. Here $\mathcal{R}$ is the set of right frame features with pixel coordinates x_i and y_i, $\mathcal{H}$ is the Hessian operator, and v_n is a step in iterative Newton solver. A pseudocode implementations of this algorithm is in Figure 2-9. The function input is a single feature's left camera and right camera tracks. The resulting output is the estimated disparity.

```
0   function estimateDisparity(left,right):
1       curve = polyFit(left.xs,left.ys, ORDER)
2       guess.x = (left.xs[0]-right.xs[0])
3       guess.y = (left.xs[0]-right.xs[0])
4       for ( i = 0; i < MAXITERATIONS; i++):
5           error = (right.ys-guess.y-polyEval(curve,right.xs-guess.x))^2
6           if (error < THRESHOLD):
7               return guess
8           gradient = computeErrorGradient(curve,right,guess)
9           hessian = computeErrorHessian(curve,right,guess)
10          guess = guess - matrixMult(matrixInv(hessian),gradient)
11  end function;
```

Figure 2-9: Synchronization Algorithm Pseudocode

2.3.4 Synchronization Results

An example of the visualization debug output of synchronization is in Figure 2-10. This allows a user to see the synchronized stereo correspondences, bottom right, and the correspondences before synchronization, top left. The green ○ represents the left

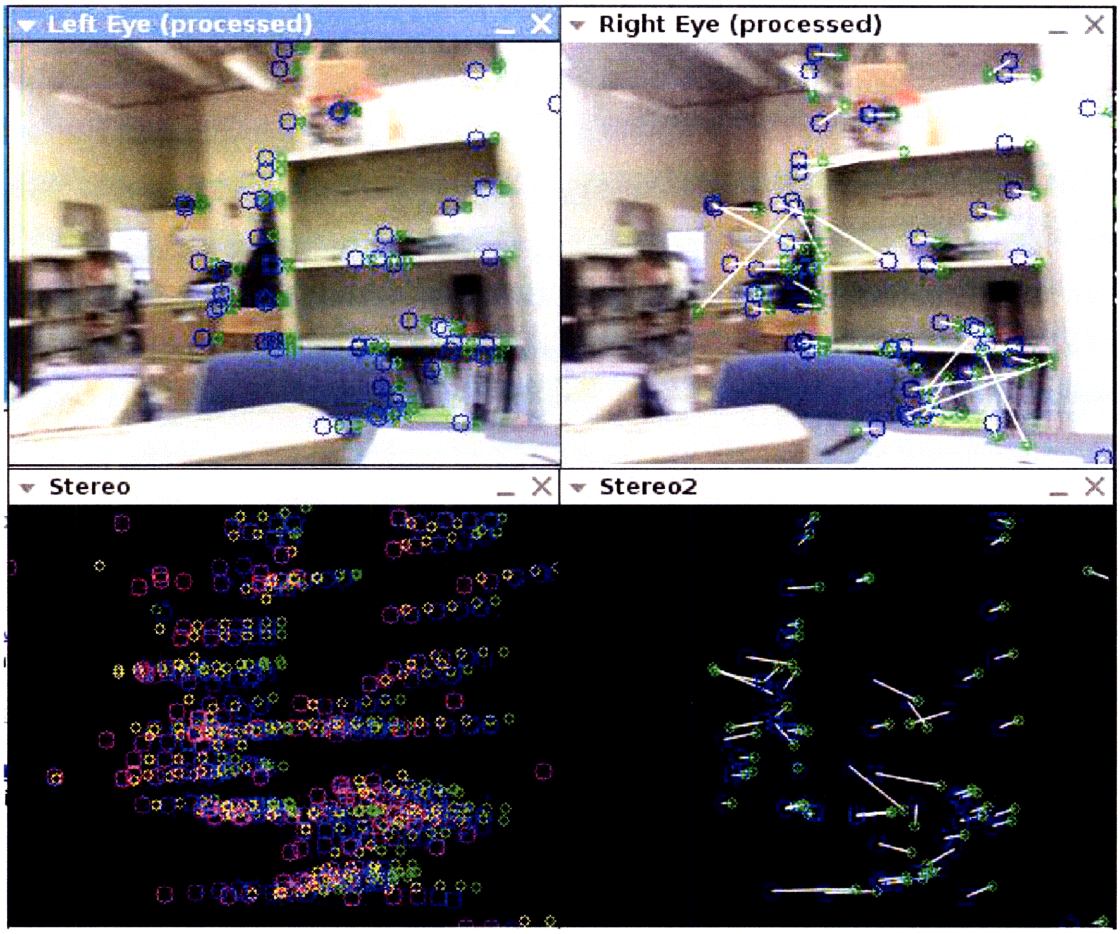
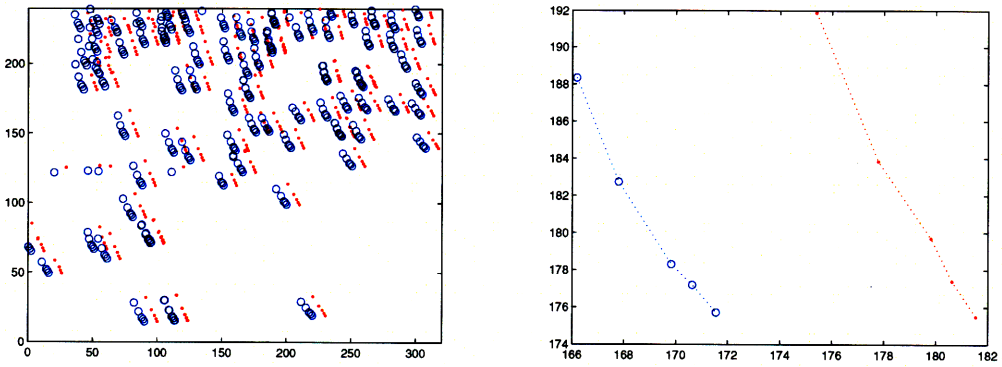


Figure 2-10: Visualization output of synchronization.

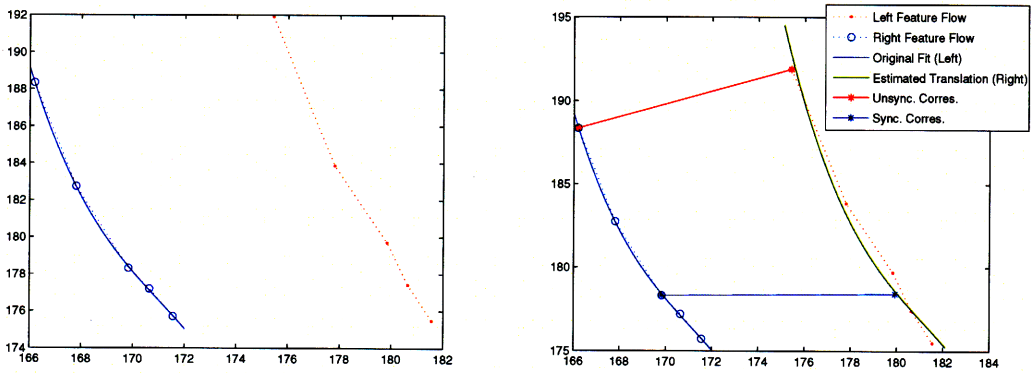
frame's features and the blue ○ represents the left frame's features. The large and small feature trails, bottom right, represent the disparities colorized for the past few frames. To see that synchronization is working properly, the user can dump the raw and corrected feature output, which would produce unsynchronized features similar to Figure 2-11 and results similar to Figure 2-12. Figure 2-11(a) shows an overall scene of unsynchronized feature tracks. Each track comprises a series of feature movements across the time window, with blue ○ representing left features and red · representing right features. The lack of synchronization is more apparent in a closeup of one of the feature's tracks in Figure 2-11(b).

The algorithm from Section 2.3.3 is applied to each feature track. For example, the left feature set of Figure 2-11(b) is first fit with a poly, shown in Figure 2-12(a). Then, the iterative Newton solver is run until the convergence of Eq. 2.5, resulting in



(a) Unsynchronized left and right feature tracks. (b) Unsynchronized corresponding feature tracks (close up).

Figure 2-11: Examples of unsynchronized features.



(a) Left Camera Image (b) Right Camera Image

Figure 2-12: Examples of synchronization performed between two features.

the translated curve in Figure 2-12(b). This leads to a synchronized correspondence for the feature, visualized by the blue line in Figure 2-12(b). For comparison, the unsynchronized correspondence is shown in a red line.

Because the unsynchronized frames simply compare the latest frame features rather than tracking the correspondence as it moves, the top right and bottom right frames have a different correspondence disparity, shown as white lines connecting the blue and green points. Because it is difficult to visualize the effectiveness of a correction in a single frame, other than that there is a difference, a typical plot of feature correspondence disparity comparing synchronization by minimizing curve error, synchronization by time, unsynchronized, and the ground truth is in Figure 2-13. In this

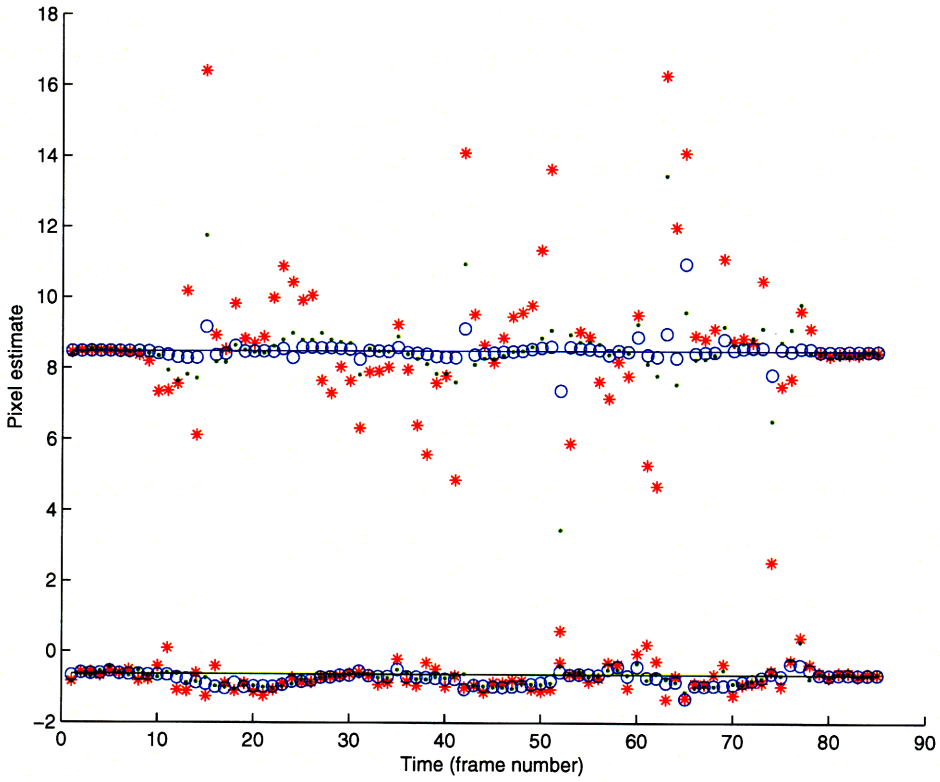


Figure 2-13: A plot detailing results for different types of synchronization. The ground truth would lie on a straight line through the steady state values of distance for each feature in the absence of motion. The point series are x (above) and y. Each point represents the detected disparity for a single feature of the cameras in random motion. The synchronization window was set to 5 frames with the middle frame as the interpolated keyframe.

plot, the data points represent disparities for a single tracked feature as the camera is quickly perturbed horizontally and vertically over 86 frames in time. The top sequence of points are the x-disparities and the bottom set points are the y-disparities. With perfect synchronization of the cameras, the detected disparity would always be the ground truth, represented by the two lines for $x = 8.47$ pixels and $y = -0.8$ pixels[2]. The red $\star$ are the unsynchronized disparities, which differ from the ground truth by up to 8 pixels in x and 3 pixels in y whenever the camera is perturbed. The green $\cdot$ are the time-synchronized disparities, which performs marginally better than

[2]These are found by observing steady-state values for the disparities in the absence of camera motion.

Table 2.2: A typical feature's max and average errors from the ground truth of the different types of synchronization.

Typical pixel error using different synchronization techniques		
Technique	Max Error	Average Error
Unsynchronized	8	4
Time Synchronization	5	2
Curve Synchronization	3	1

the unsynchronized version, exhibiting a maximum difference of 5 pixels in x and 1.5 pixels in y. The blue $\circ$ are the curve-synchronized disparities, which differs by a maximum of 3 pixels in x and less than 1 pixel in y. A summary of these results is seen in Table 2.2.

The effect of this synchronization is more pronounced when the data is used for stereo location and mapping. By synchronizing the frames, a stereo correspondence for each feature will have more consistent disparities in the presence of camera motion. This results in more consistency in the location estimate for the feature and results in improved mapping, which is demonstrated in Chapter 4.

2.4 Correspondence Location Determination

After finding synchronized stereo correspondences with respect to the camera, it is important to estimate the location of this feature based on the disparity of the correspondence. This estimate can be continuously updated as more measurements are made of the feature in time. To estimate the location of the feature, two rays are created for each of the frames. Each ray has the form:

$$\mathbf{r} = \left[p_x - \frac{\mathrm{X_{RES}}}{2}, \quad p_y - \frac{\mathrm{Y_{RES}}}{2}, f \right] \tag{2.7}$$

where, f is the focal length of the lens in terms of pixels, which can be calculated using the pinhole camera model and trigonometry in Figure 2-14 to be $f = \frac{\mathrm{X_{RES}}}{\arctan \frac{\mathrm{fov}_x}{2}}$. $\mathrm{Y_{RES}}$ and $\mathrm{X_{RES}}$ are the resolutions in the y and x directions, p is the pixel location of

the feature and r is the resulting ray. Since two arbitrary rays in 3D do not necessarily cross, an estimator is used to find the point closest to both rays. This estimator is derived in [3]. It states that:

$$\tilde{\mathbf{r}} = \mathcal{A}^{-1}\mathbf{b} \tag{2.8}$$

where

$$\mathcal{A} = \sum_{i \in \text{Features}} w_i \left(I - \hat{\mathbf{r}}_i \hat{\mathbf{r}}_i^T\right) \tag{2.9}$$

and

$$\mathbf{b} = \sum_{i \in \text{Features}} w_i \left(I - \hat{\mathbf{r}}_i \hat{\mathbf{r}}_i^T\right) \mathbf{l}_i \tag{2.10}$$

Here, $\tilde{\mathbf{r}}$ is the estimated feature's location relative to the camera, w_i is an arbitrary weight assigned to how much confidence there is in a measurement, $\hat{\mathbf{r}}$ is a normalized unit vector of a ray found using equation 2.7. Finally, $\mathbf{l}$ is the location that made the observation. This is set to $[0, 0, 0]$ for the left camera and $-T$ from equation 2.1 for the right camera's viewpoint. The form of this equation lends itself to continuous measurement and estimation while only occupying $O(1)$ storage space. By keeping track of only $\mathbf{b}$ and $\mathcal{A}$ in this least squares minimizer, repeated estimates of a feature's location can be made by adding them to the matrices. In Chapter 4, the location and measurement are transformed into a global reference frame when the mapping module is used along with location information.

2.5 Image Processing Summary

The images were acquired through the use of an `HTTP_GET`, `multipart/x-mixed-replace` image stream and processed for features using OpenCV's `cvGoodFeaturesToTrack` and `cvCalcOpticalFlowPyrLK`. The resulting features for the right camera were placed into the left camera frame through rotation and a translation, which were extracted as extrinsic properties of the stereo configuration using the MATLAB camera calibration toolkit. Next, two feature synchronization techniques were used to estimate a right feature location based on the feature's optical flow history within a

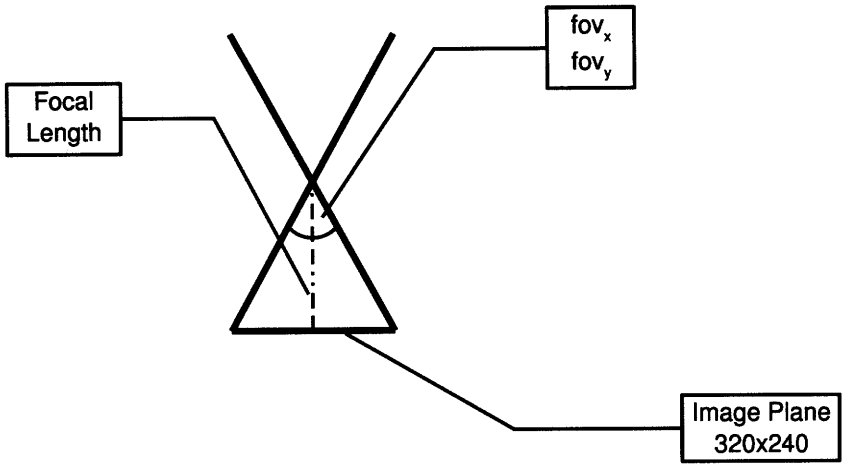


Figure 2-14: Trigonometry calculation leading to $f = \frac{X_{RES}}{\arctan \frac{fov_x}{2}}$

window of $N = 5$ frames. The optimal synchronization technique, curve-matching as described in Section 2.3.3, is used in this project to make stereo correspondence with unsynchronized cameras possible as demonstrated in Chapter 4.

Chapter 3

Object Detection

A useful map of the environment should not only contain features from various objects detected in the environment but should provide the locations of *objects of interest*, which match some mission-specific criteria. Examples of *objects of interest* include vehicles for reconnaissance missions or people for surveillance missions. For this thesis, a simple feature clustering detector and the OpenCV implementation of a Haar transform-based classifier object detector detailed in [21] was used. These methods are implemented as object detection modules for the system and can be enabled or disabled depending on the needs of a mission. Since the Haar feature classifiers used to find target objects in an image needs to be pre-computed from training sets of positive and negative samples, a natural limitation of this type of detection is that it requires the general shape of *objects of interest* to be known prior to a mission. Once objects are detected, they are passed to the mapper as special features with their centroids as locations so that they can be added to the map. To assist mission planners in creating a classifier for use in the stereo mapping system, an object specification and classifier testing frontend was created. A diagram of how this relates to the entire system is given in Figure 3-1.

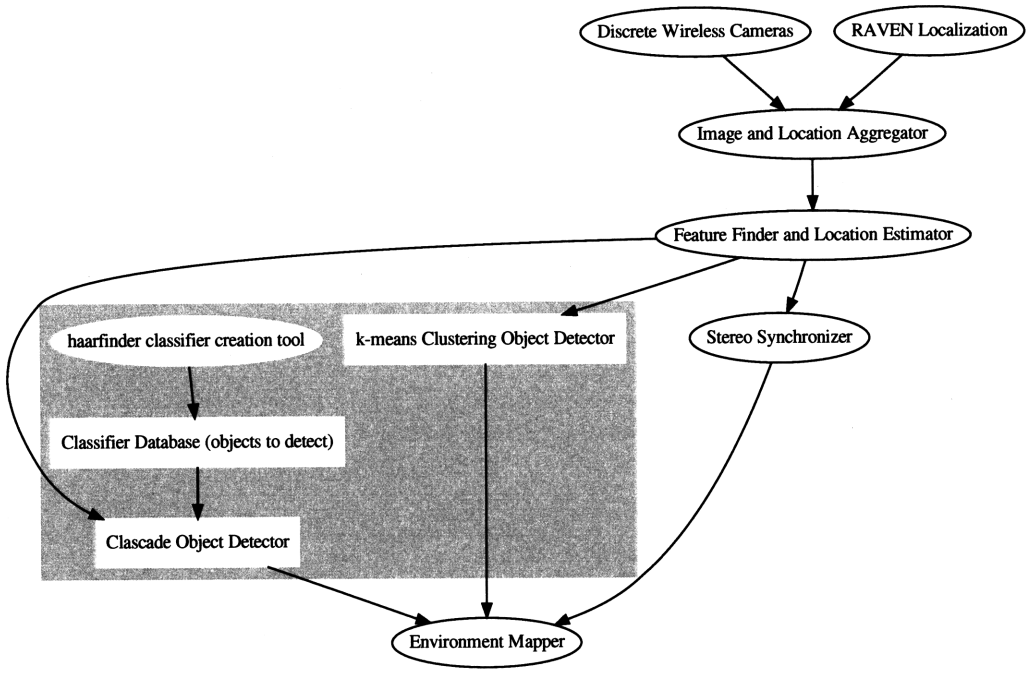


Figure 3-1: System components of the object detection system.

3.1 Clustering-based detection

A general clustering identifier module highlights potential objects through the use of k-means clustering and a proximity threshold. It operates on a set of features used in one stereo correspondence. K-means was chosen as a clustering algorithm for its fast convergence compared to more deterministic algorithms such as the quality threshold clustering algorithm developed in [18]. For the cluster detector modules, k is set to 20% of the total number of good features found in a set of stereo correspondence frames with the idea being that at least 5 features are needed to determine an object based on feature correspondences. The k-means algorithm is then run among the points using OpenCV's k-means implementation, `cvKMeans2`. After clustering, each cluster's three-dimensional covariance is used as the threshold to determine what is a positive detection. The three-dimensional covariance is also used as a metric for the *size* of the object, as distinguished in visualization by a larger circle.

3.1.1 Typical Cluster Output and Visualization

The result of a typical clustering run on a scene is in Figure 3-2. In this figure, the top left presents a current shot of the left camera's video stream with current features represented by green ○ and previous features represented by blue ○. The top right image shows the stereo correspondence with synchronization. This image has left camera features represented by green ○ and right camera features by blue ○. Corresponding features are linked by a white line. The results of localization from the stereo disparity in this image are displayed in the bottom image, which shows them in 3D. Note that features from the back of the room are correspondingly further away. Finally, the top center image shows the results of the k-means clustering, which is performed on the image features from the top left image.

The large dots in the k-means image is the centroid of the features that support it, while the small dots are the features. Each cluster is visualized by having a different color. After dividing into clusters, the algorithm determines which of the clusters are objects by comparing a threshold with the covariance of the objects in 3D. Essentially this covariance is calculated using the 3D data that was used to produce the bottom image. The covariance also determines the size of the dots in the visualization. In this instance, the clustering has determined that the conduits on the wall, the student in the chair, the bicycle, the desk, and the hanging plane are all objects. The detected *object of interest* can be requested when querying the environment map. It should be noted that while this thesis does not attempt to solve the full SLAM problem, these clustered features are ideal landmarks for use should localization algorithm require a list of currently existing landmarks from its map.

3.2 Classifier-based Detection

Mission planners sometimes need to detect specific objects rather than generic objects in the scene. OpenCV's Haar transform-based classifier detector, implemented as the function `cvHaarDetectObjects`, provides one solution to detecting specific objects. The contribution of this project, besides using the detector in the image processing

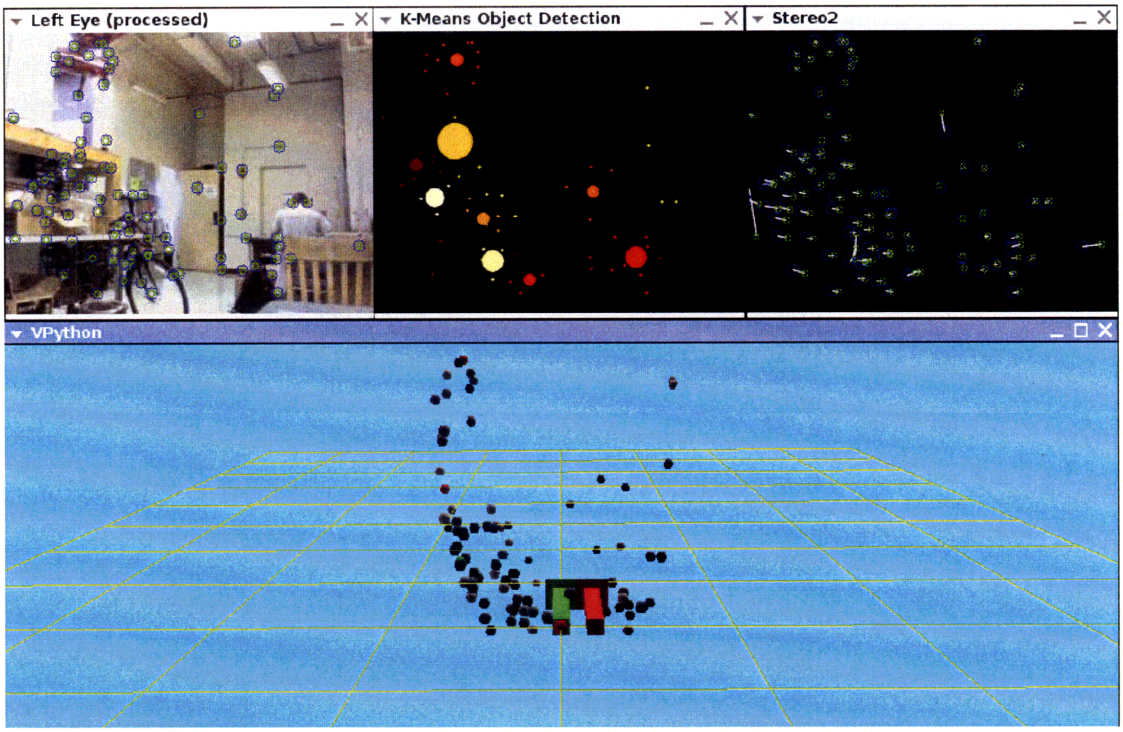


Figure 3-2: k-means clustering and object detection with associated windows of stereo correspondence, feature points, and a 3D map of the features.

pipeline, was the creation of a set of software tools, `haarfinder`, which takes as input a set of background images, a set of foreground images. `haarfinder` simplifies the process of specifying bounding boxes by iterating through the foreground images and providing a drag and drop interface. Additionally, it provides a frontend to OpenCV's `opencv-createsamples`, `opencv-haartraining`, and `opencv-performance` to allow easy tweaking of tolerances and variables, which is important because these adjustments allow a mission planner to target a specific probability of detection (P_D) versus probability of false alarm (P_{FA}). It also allows the planner to visualize tests of the created classifier against sets of images. A screenshot of the graphical user interface for this tool is in Figure 3-3.

In this screenshot, one of many three-forth views of the truck is being used to train a classifier set. The user can use this tool to specify bounding boxes and delete old bounding boxes for a large set of images in a short amount of time. Because of how classifiers are formed in creation, it is essentially impossible to debug a classifier

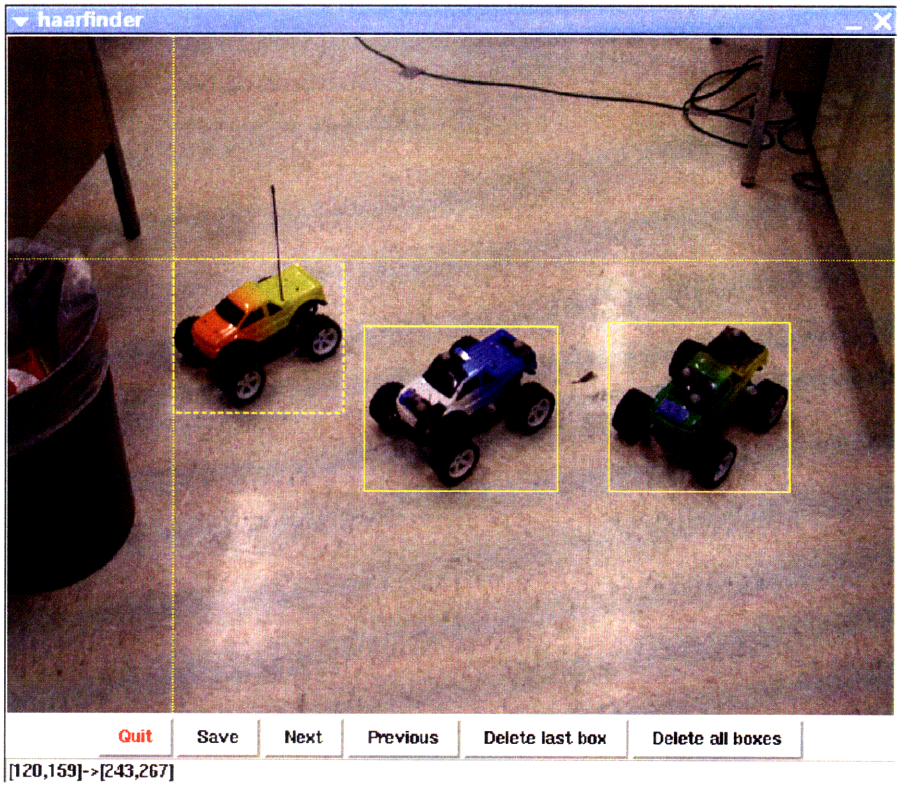


Figure 3-3: The graphical user interface of `haarfinder`.

specification file, which is a set of weights describing arbitrary Haar features and configurations. Instead, training parameters can be modified, usually trading off of P_D and P_{FA}. This kind of trial and error necessitates this type of tool, which speeds up the process of iteration.

When a classifier object detection is turned on, the vision pipeline processes every left frame through OpenCV's `cvHaarDetectObjects` using a pre-specified classifier that is loaded into memory. The left frame is used because finding the location of a detected object involves a simple look-up of the location corresponding to the detected object without preceding the look-up with a transformation. This is because the left frame is the starting point when stereo synchronization is used to estimate where corresponding features in the right frame would be.

Any objects detected by this classifier will be highlighted in the visual interface and accessible through the map. The location of the detected object is provided by retrieving the location of any features enclosed by the boundaries of the object.

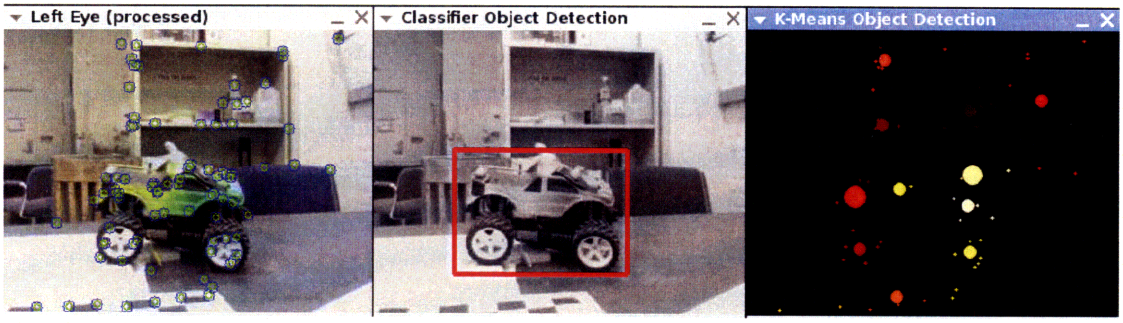


Figure 3-4: An example of the classifier trained to recognize small ground vehicle.

3.2.1 Classifier Visualization Example

An example of a classifier running next to a clustering object detector is in Figure 3-4. In this figure, the classifier has been trained on 50 positive side-view images of the truck and 5000 background images, consisting of random photographs not containing the side of the truck. Additionally, the training was completed with default parameters except for `nstages`, a parameter from [19], which was set to 7 from the default of 14 to improve performance. The left image is the left frame's features and the right image is the clustering detector, identical to the visualization in Sec. 3.1.1. The center image contains the training vehicle with a bounding box, which is drawn after the classifier detection returns a found vehicle. A discussion of the effectiveness of this method is given in Sec. 3.3.

3.3 Detector Performance

The performance of the clustering object detector is evaluated by taking images of scenes with planted objects and determining how often these objects that are found by the clustering system. Similarly, the classifier-based detector, trained as mentioned in Sec. 3.2.1, is used on the same set of images. The performance of the clustering detector, the classifier detector in proper orientation and the classifier detector offset by approximately 10° are then compared using their rate of detection and rate of false positives. Because the scenes used in this test are artificially populated by

objects of interest and miscellaneous objects and the detectors have different criteria for detection, the following definitions of detection and false positive are used.

3.3.1 Criteria for Detection and False Positive

A k-means detection is defined as finding any object's location within a frame, while a k-means false positive is defined as the detection of a non-object, such as a color deformation or points from edge discontinuities. A classifier detection is defined as the identification of the target object's location in a frame, with a false positive being the tagging of anything else other than the target object. Using these definitions, a set of 100 images similar to the images in Figures 3-4, 3-5, and 3-6 were tested with both detectors and the probabilities for detection and false alarm recorded.

3.3.2 Test Results

The 100 images were taken from the stereo camera in a relatively sparse background with exactly 50 images containing at least 1 target object and the other 50 images containing no target objects but at least 1 non-target object. Here, a target object is the object that the classifier has been trained to recognize. The results are presented in Table 3.1. The clustering algorithm located the target objects in all but 4 instances. In all 4 failed detections, the object in question was positioned very far from the camera. In 14 cases, the clustering algorithm detected a object where there was no object because of the feature tracker picking up discolorations in the background that were in close proximity to each other.

Table 3.1: Probabilities of detection and false alarm for detectors

Technique	P_D	P_{FA}
Clustering	0.96	0.14
Classifier (Side View)	0.82	0.19
Classifier ($\sim 10°$ skew)	0.67	0.18

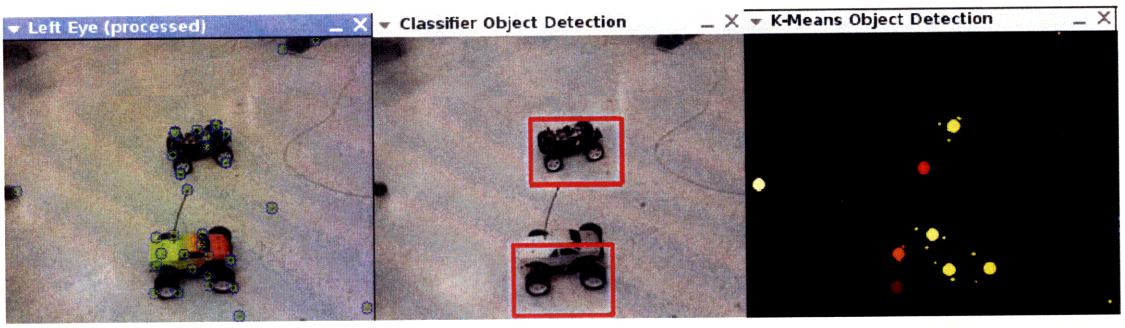
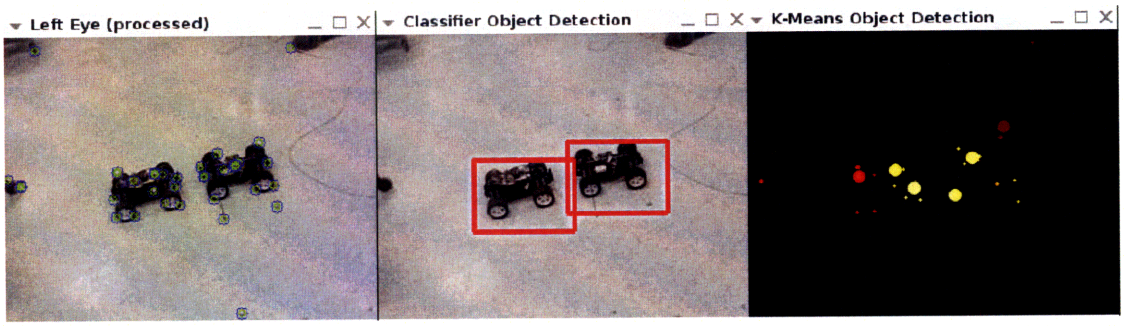


Figure 3-5: An example of the classifier trained to recognize ground vehicles.

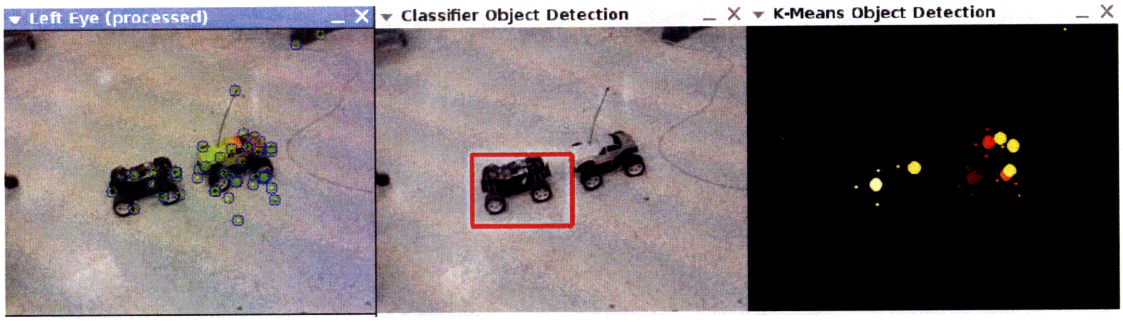
3.3.3 Discussion of Results

Some typical examples of detection and non-detection are shown in Figures 3-5 – 3-7. Again, positive detections by the classifier are boxed in red. The first example, Figure 3-5, is a typical detection by the classifier detector. Note that the clustering detector finds multiple objects where one object exists. The reason for this error is that the algorithm makes no attempt to run k-means for multiple k until a specified average covariance threshold is reached. This design decision was made for computational efficiency.

Since the classifier was only trained to recognize an object in a certain pose, differently posed target objects are not counted against the detection rate. One example of when the classifier did not detect a target is in Figure 3-6, where Figures 3-6(a) and 3-6(b) present similar configurations of vehicles. Figure 3-6(b) does not contain a detection on the right vehicle, where a top has been attached to the vehicle, which has also been angled upwards. Based on the ability of the detector to find vehicles with tops in Figure 3-5, the change in angle is the most likely reason for the lack of detection. Support for this hypothesis is seen in Figure 3-7. Not surprisingly, the front-facing vehicle is not identified by the classifier trained to recognize the side of the vehicle in Figure 3-7(a). Additionally, Figure 3-7(b) presents a more top-down pose, where the wheels appear significantly more elliptical than in the side views used for training. For all of these examples, the clustering detector has no problem finding the image's objects. The locations of objects found by the clustering detector are indicated by one or more circles over the corresponding locations of the vehicles

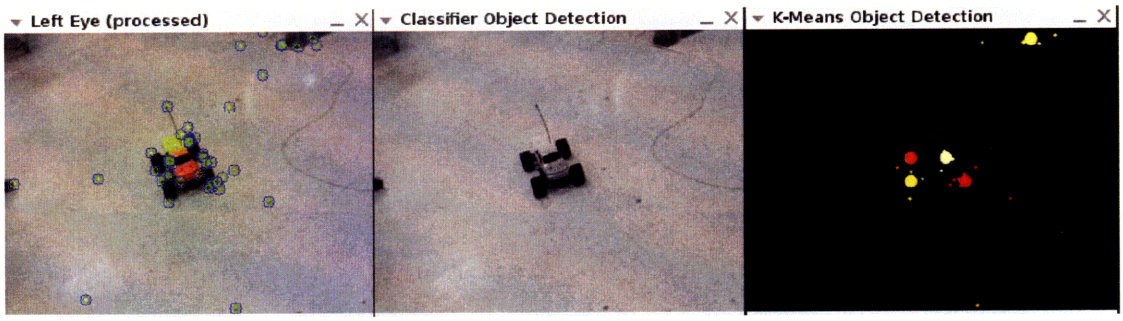


(a) Positive recognition

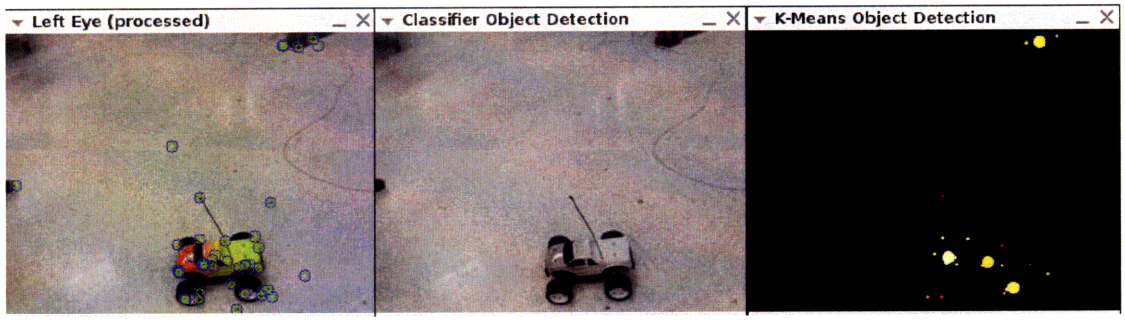


(b) Negative recognition

Figure 3-6: An example of how lighting can affect classifier performance.



(a) Differences in object pose cause classifier failure



(b) Slight pose differences can also affect performance

Figure 3-7: Pose can affect classifier performance. The objects are still detected by clustering.

in the k-means object detection windows. These results reinforce the idea that the clustering detector works on very general objects and the classifier detector identifies a specific pose of an object.

3.4 Chapter Summary

The two object detection modules in this chapter allow control systems to detect specific objects or objects in general using classifiers or clustering. These modules can be selectively enabled in the image processing and stereo mapping pipeline, allowing system developers to identify landmarks and use generalized objects in their planning algorithms. The output of this system is queried against the localization out from Chapters 2 and 4 to determine the location of the detected objects. Results show that the clustering module detects objects with a 0.96 probability, only missing objects that are very far from the camera. Additionally, the classifiers will detect specific objects in poses that it was trained for with 0.82 probability and 0.67 probability in the presence of a 10° skew. The implication of these results is that mission planners can reliably use clustering to find `objects of interest` and they must train classifiers for multiple poses if they need to detect specific objects in multiple orientations. Once multiple poses of an object are trained, the detection rate will increase.

Chapter 4

Environment Map Generation

The synchronized stereo, feature location, and object detection all contribute to an environment map created as the last stage in this thesis's processing pipeline. This relation to the rest of the system is shown in Figure 4-1. From the SLAM literature [10, 25], several ways of storing a map have been suggested, with the main tradeoffs being memory usage and computational speed. While this thesis does not seek to implement a full SLAM solution, it is designed to create a map that may be used to run and test localization algorithms. Thus a method of storing the map and providing access to its contents was needed.

4.1 Map Data Structure Selection

The map representation was chosen to be an occupation grid, similar in concept to the one used in [29]. It is implemented using a hashtable with linked lists and is visualized in Figure 4-2. This data storage mechanism was chosen over large lists of points to trade more storage space for faster access in the presence of many data points. The lookup for the third direction uses a linked list rather than a hashtable. While hashtables have $O(1)$ lookup time versus $O(n)$ linked lists, the tradeoff is made feasible by the relatively small number of points in the third direction. The hashtable of hashtable approach would also have a higher time and memory cost when creating a secondary hashtable instead of a linked list for the primary hashtable's entries. In

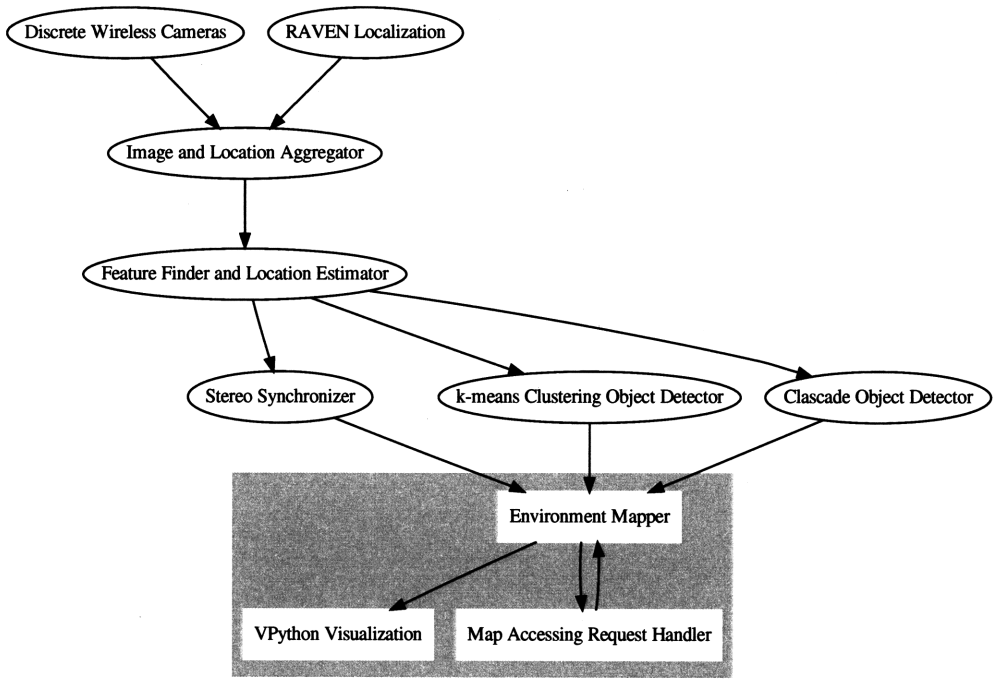


Figure 4-1: System components of the mapping system.

addition, using linked lists reduces the number of hash look-ups required from $\frac{E}{D}^3$ to $\frac{E}{D}^2$. Here, E is the amount of error range to look for a previously present feature and D is the size of the grids in the hash table. For this project, D was chosen to be 10 cm and E to be 30 cm. The choice of these parameters was made after experimentation showed that they worked better to find previously located features.

4.1.1 Data Structure Performance Comparison

A comparison of the performance of the three suggested implementations was found through simulation in Table 4.1. For this simulation, N random points were generated and placed into each of the three data structures. The time it took for each insertion,$\overline{T_I}$, was recorded along with the size of the resulting structure, S. To simulate looking up points that may or may not be new, the simulation queried the data structures with a $p = .5$ probability of the queried point coming from the pool of points known to be in the data structure. This simulation measures an average

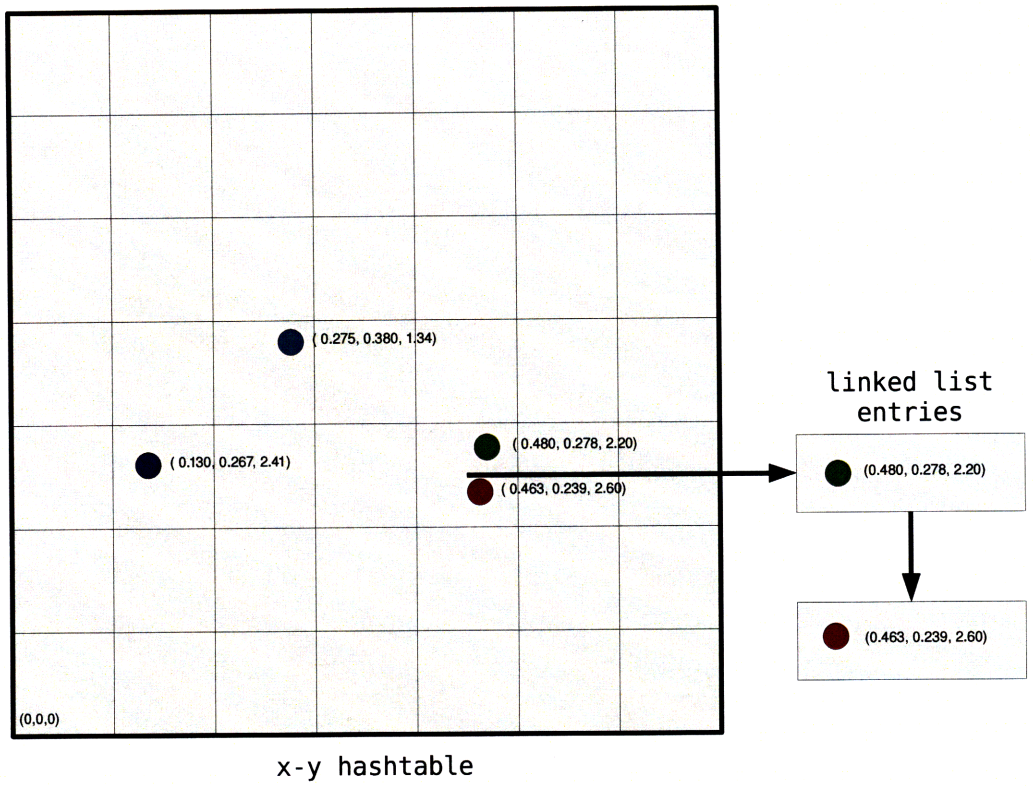


Figure 4-2: The occupation grid is stored as a hashtable keyed with rounded x-y for grid location and entries of linked lists of features in each grid.

lookup time of $\overline{T_L}$. These results are consistent with the design goals of getting fast lookup and insertion times when over 10,000 data points are used. An average lookup time for linked lists was not established because the lookup did not finish within 30 minutes of starting the simulation.

4.1.2 Data Structure Usage

To determine a grid location for a point, rounded x and y coordinates are used as the key. Consequently, features with relatively the same projection on the x-y plane are stored in the same occupancy grid in the same linked list. The linked lists are used to minimize the size of the hashtable while maintaining fast lookups in the x-y plane, which is more important if the mission space is large in the x-y than in z.

This is true for autonomous robot missions such as reconnaissance, where a vehicle will monitor a significant x-y plane area than altitude z. Coincidentally, the indoor RAVEN development room is larger in the x-y dimension as well. If a mission is encountered which is larger in the vertical dimension, the hashing key can easily be changed to use x-z or y-z with the linked list in y or x, respectively.

In addition to the coordinate-lookup hashtable and linked list, the mapper contains a unique id-keyed hashtable for features, which can be used to update estimates of feature locations without looking for the feature in the map occupancy grid. This provides a theoretical increase in performance with a $O(1)$ lookup versus $9O(1)O(n) = O(n)$ lookups and list traversals, assuming near constant-time hash lookups and $O(n)$ linked list lookups [6].

4.2 Processing Correspondence Data

The system can receive location data using the lightweight communications and marshalling (LCM) protocol [30]. These LCM packets typically come from localization algorithms or from RAVEN and its Vicon system. If the location data is available, the l_i location inputs to the feature location estimator, Eq. 2.10 of Section 2.4, are first translated into the global coordinate system. Additionally, the rays generated through feature observation are placed into the global frame through a rotation by

Table 4.1: Map storage method performance

Map Storage	S (MB)	$\overline{T_I}$ (μs)	$\overline{T_L}$ (μs)	N
Hashtable with Linked Lists	1.3	7.8	5.2	10,000
Hashtable with Hashtables	3.0	9.1	7.2	10,000
Linked List	0.8	1.4	483	10,000
Hashtable with Linked Lists	8.6	5.8	6.4	100,000
Hashtable with Hashtables	17.8	10.9	7.0	100,000
Linked List	7.7	2.8	DNF	100,000
Hashtable with Linked Lists	77.5	6.7	14.7	1,000,000
Hashtable with Hashtables	131.1	20.9	9.2	1,000,000
Linked List	76.5	3.9	DNF	1,000,000

the camera orientation. For Vicon data, which provides the location as a vector and orientation as a quaternion, the translation is offset by the location with swapped $-y$ and z coordinates because the y-axis points down in the camera frame and z points outward from the camera sensor through its lens. The rotation is handled by an axis-angle rotation as a function of the quaternion in Eq. 4.2. The quaternion is converted into axis-angle form and VPython's [43] axis-angle vector rotation function is used to rotate all measurements before they are inserted into the estimator, Eq. 2.7 of Section 2.4.

$$r_{\text{axis}} = \left[\frac{q_x}{\sqrt{1-q_w^2}}, \frac{q_y}{\sqrt{1-q_w^2}}, \frac{q_z}{\sqrt{1-q_w^2}}\right]^T \tag{4.1}$$

$$r_\theta = 2\arccos q_w \tag{4.2}$$

Here, r_{axis} is the rotation axis, r_θ is the rotation angle, and q_x,q_y,q_z,q_w are the components of the quaternion. By inserting the camera location data into the estimators of Eq. 2.8 along with the output of the axis-angle rotation from Eq. 4.2, the features' locations are computed at their global locations rather than in the relative camera frame. An abbreviated implementation of this is described in Appendix A.

4.3 Map Access for Other Applications

The feature map is designed to be accessed by other modules. This is accomplished through a series of accessor functions that will return map data at requested locations or the entire map. The objects detected from Chapter 3 can also be requested. To make the map accessible to localization algorithms or other applications outside of this framework or the network, the map application listens for LCM requests for information and replies with the information using LCM.

To provide users with a method of debugging their classifiers without writing an application that accesses the map through its accessor functions, a 3D visualization tool using Visual Python [43] was modified for the purpose. The interface for the visualization tool is in the bottom half of Figure 3-2. The tool provides a 3D map

with the location of the camera, the location of any estimated features, and the location of any objects. It is based on the `3dvis` application from RAVEN, developed in [3], and is essentially a task-specific version that contains extended functionality for storing a map even when the actual visualization component is deactivated.

4.4 Map Performance

Because this thesis does not implement a full SLAM solution, complete with more sophisticated location estimators, feature locations are based on location information provided by RAVEN and by least squares estimators described in Section 2.4, which approximate feature locations over multiple stereo measurements. The performance of the mapping aspect of this thesis will be comparing the maps created from synchronized stereo vision and from unsynchronized stereo vision. Additionally, the performance is compared to a map of the area created through the use of single-camera SLAM with inertial data, simulated by VICON. All maps are created from images and data gathered by moving a camera through the RAVEN environment in the span of approximately 2 minutes and processed in real time. For consistency, the archiver and replayer features of the vision system, detailed in Sec. A-6, were used to run the various algorithms on the same data.

4.4.1 Synchronized Versus Unsynchronized Maps

First, the performance of synchronized versus unsynchronized stereo mapping is compared. The difference between the methods is that the unsynchronized map, Figures 4-4(b) and 4-5(b), was created corresponding the latest received right and left frames while the map in Figures 4-4(a) and 4-5(a) was created using correspondences of the left frame's features with synchronized features output by the curve-synchronization described in Section 2.3.3.

Each of these graphs contains an overlay of the approximate layout of the room based on rough measurements, with map points in blue and the room element overlays in red. The overlays are the left wall, back wall, right wall, right counter, right desk,

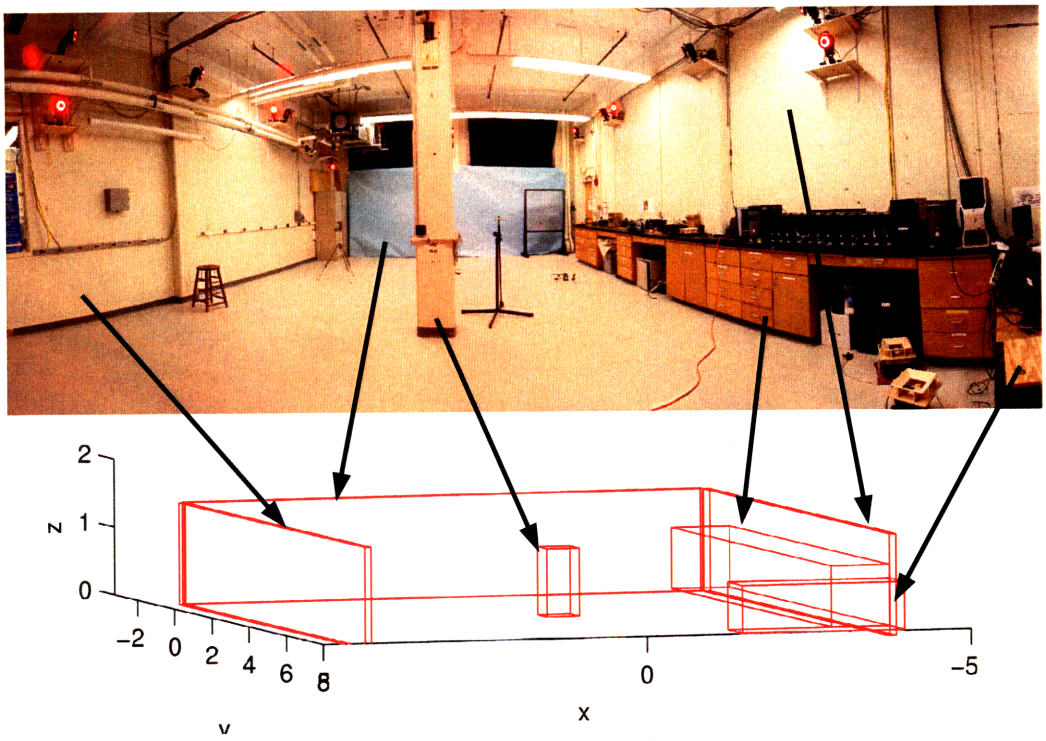
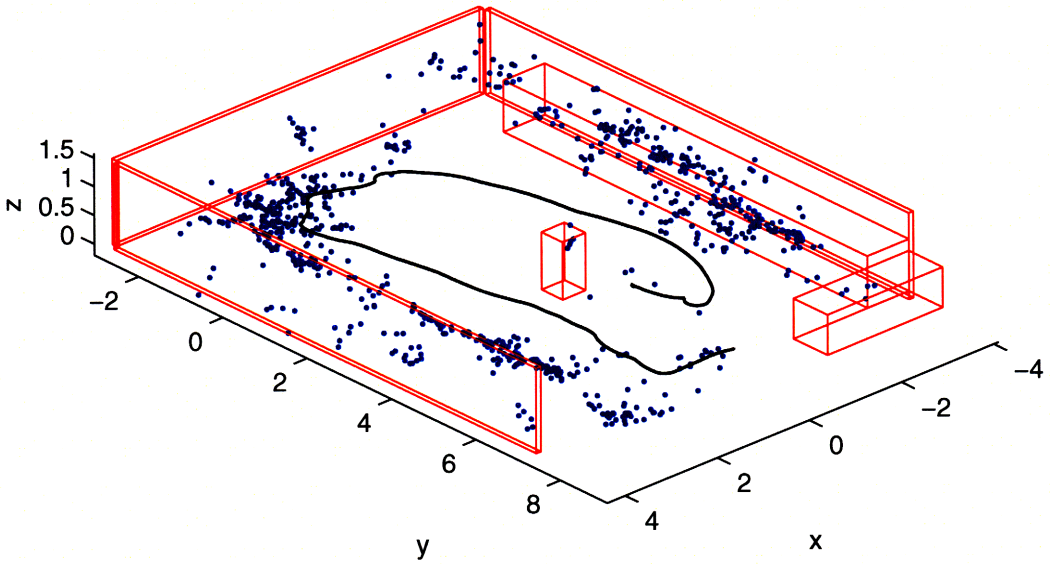


Figure 4-3: Environment being mapped

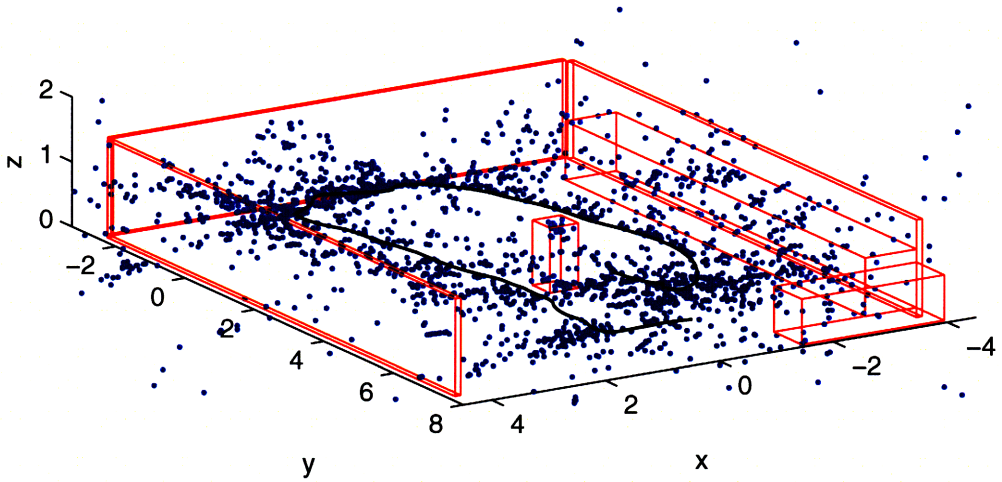
and center column. They correspond to the elements in the actual room according to Figure 4-3. The overlay does not include certain elements that were present in the room, such as the back left storage cabinet and tripod, which correspond to the multiple features located in that area of the room. The black line in each graph marks the route of the camera. The extreme noise in the estimated feature locations seen in Figures 4-4(b) and 4-5(b) indicate poor stereo performance, while the relative correspondence of estimated locations to measured locations indicate much better performance in Figures 4-4(a) and 4-5(a). The synchronized measurement error was approximated to be less than 8 percent using known wall locations and the location from where a measurement is taken.

4.4.2 Synchronized Stereo Versus Monocular Map

The synchronized map from the previous section is compared to a map created using a single camera in Figures 4-6 and 4-7. Here, the single camera map features are

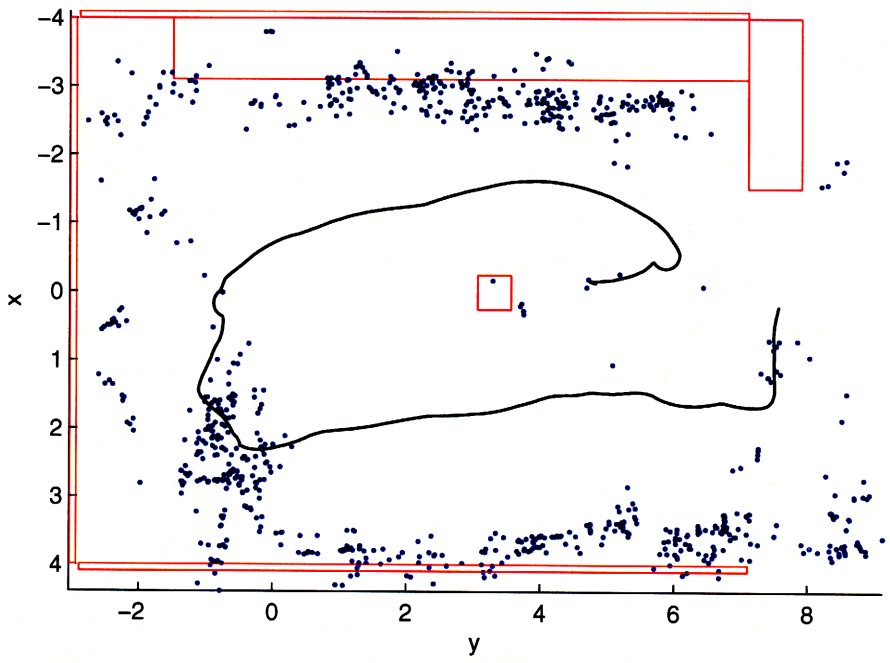


(a) Synchronized map of the RAVEN room, 3D.

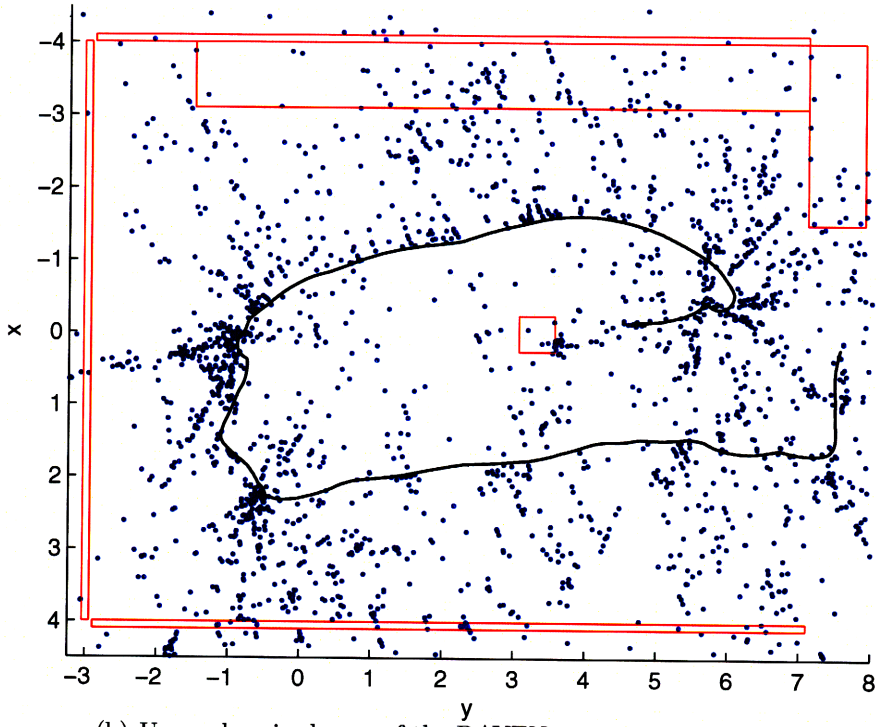


(b) Unsynchronized map of the RAVEN room, 3D.

Figure 4-4: Three dimensional views of the maps created using unsynchronized and synchronized cameras.



(a) Synchronized map of the RAVEN room, top-down view.



(b) Unsynchronized map of the RAVEN room, top-down view.

Figure 4-5: Top-down views of the maps that have been projected on the x-y plane.

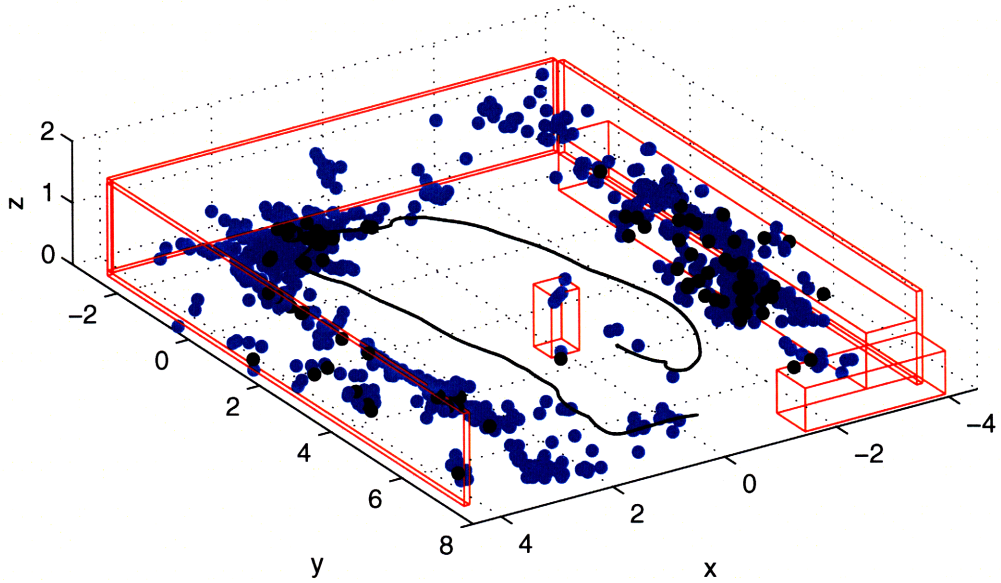
represented by black • and the stereo results are represented by blue •. The single camera mapping used absolute VICON location data and an estimator similar to the one used by Eq. 2.8. This data and implementation was run using the ACL's localization project [2]. The single camera map did not use a localization algorithm because of the nature of the camera walk-through, a ground movement rather than an autonomous helicopter, which causes extreme errors in the state estimate used by [2] to perform mapping. While this map does not capture the true performance of [2], it does represent the absolute best that [2] could perform in the absence of state estimation error. Thus, it is a fair comparison with the synchronized stereo data, which was also created using VICON localization data.

An analysis of the map with both monocular and stereo data in Figure 4-6 reveals that the stereo mapper localized significantly more data points than [2]. Nearly all of the points that are found in the monocular map have a nearby stereo point, showing that the stereo system identifies all the points that the monocular system identifies and more. The monocular system identifies 327 unique points, while the stereo system identifies 881 unique points.

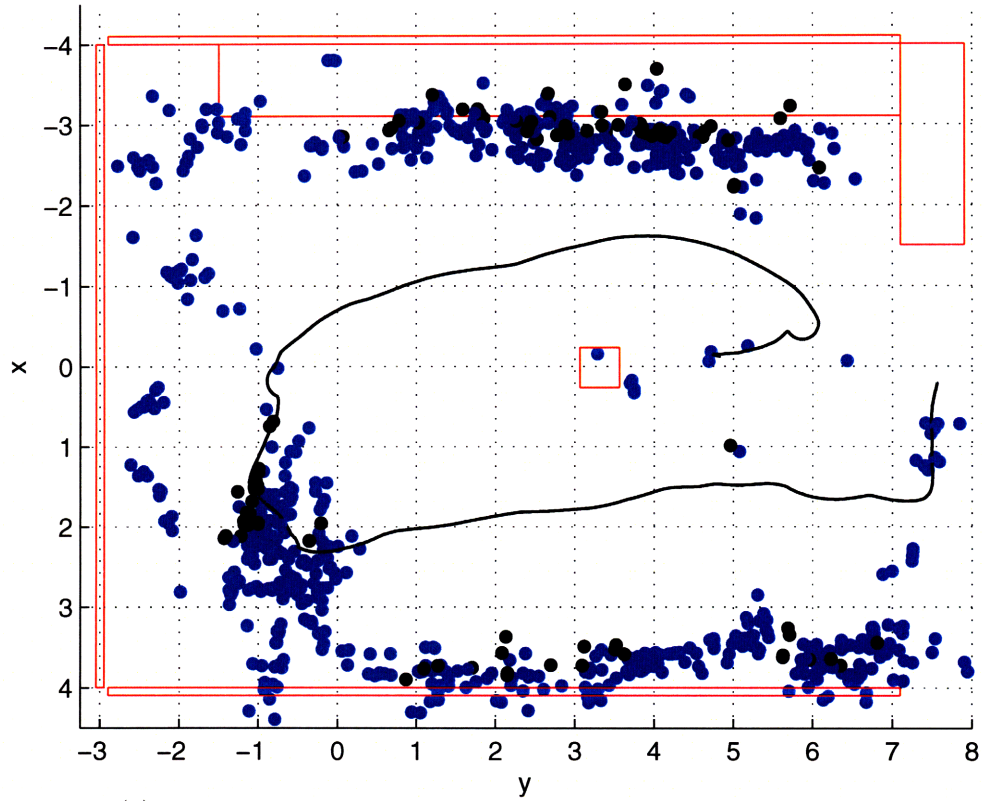
A second test involving a different movement pattern, with the camera facing inwards towards the column rather than outwards at the walls produces the maps in Figure 4-7. The results of this test show that not only did the stereo approach capture more points on the center column, it manages to locate points around the room despite most of the room being occluded by the column during the test. This is significantly better than the single camera approach, which makes no feature detections other than the column. In terms of unique points tracked, the single camera system found 169 points versus the stereo system's 268 points.

4.5 Chapter Summary

An environment mapper was implemented to create maps given location data for the camera. The mapper sends the location data to the feature location estimation algorithms from Section 2.4, whose corresponding output is added to the map. The

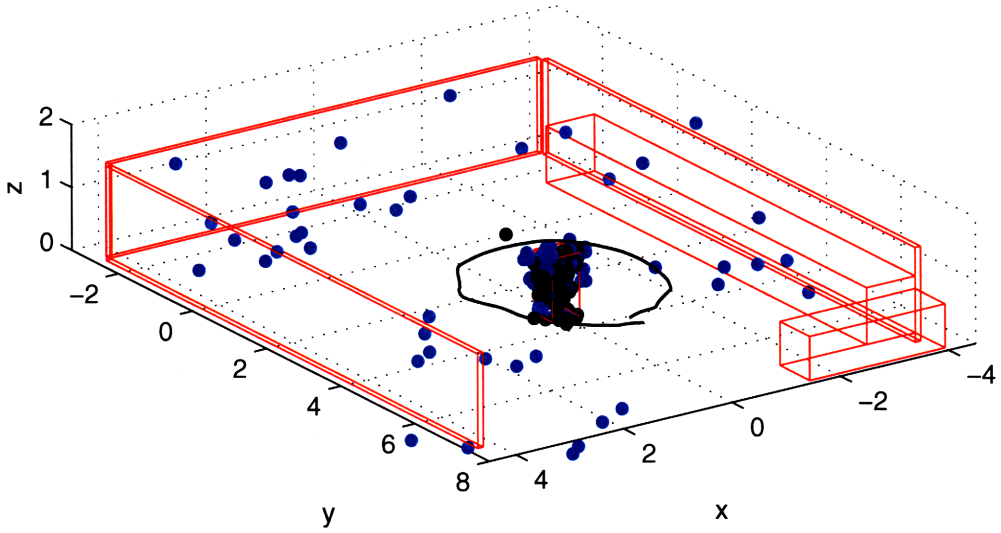


(a) Synchronized map overlaid onto monocular map, 3D.

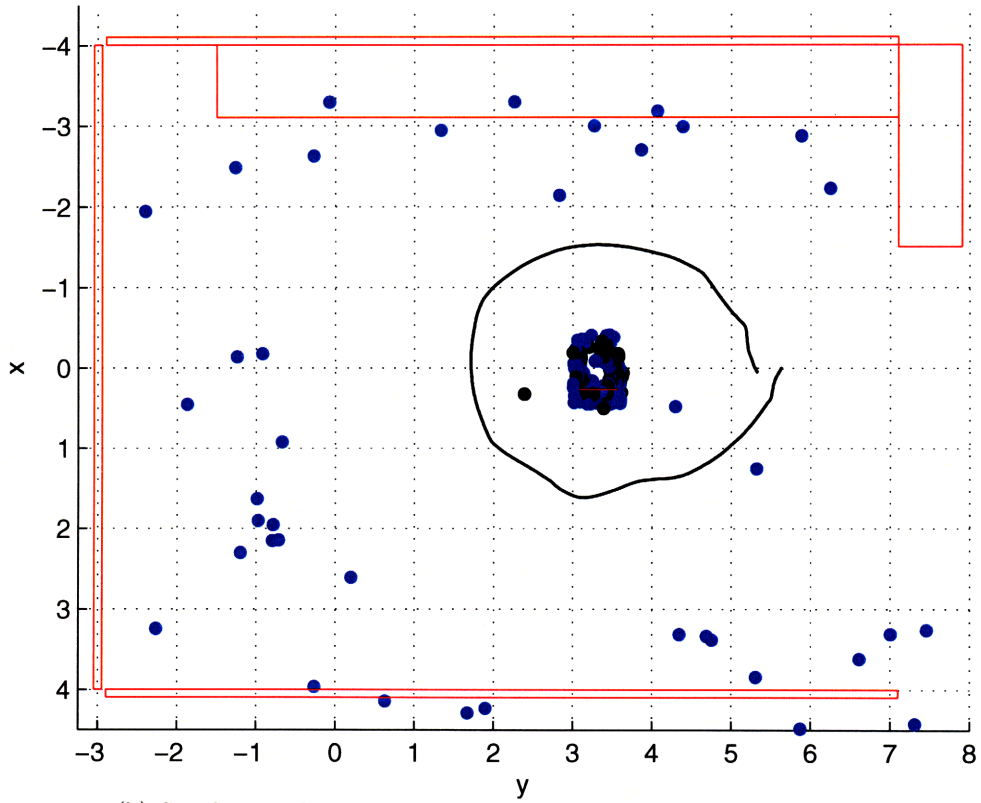


(b) Synchronized map overlaid onto monocular map, top-down view.

Figure 4-6: Synchronized map overlaid onto monocular map, constructed using truth data.



(a) Synchronized map overlaid onto monocular map, 3D.



(b) Synchronized map overlaid onto monocular map, top-down view.

Figure 4-7: Synchronized map overlaid onto monocular map, constructed using truth data.

map stores points in an occupancy grid, implemented using a hashtable with rounded x-y coordinates for keys and linked lists of point objects for entries.

This mapper was used to test the performance of the camera synchronization from Section 2.3.3. Results indicate that mapping feature locations estimated with synchronized stereo vision produces a significantly more useful map than using stereo vision without synchronization. Additionally, in a comparison between the synchronized stereo and a single camera system, both running in real time, the stereo system created a map with more features and mapped portions of the environment missed by the single camera.

Chapter 5

Conclusions

This thesis describes the design, implementation and evaluation of a modular stereo vision and mapping system that tolerates asynchronous cameras. This ability allows COTS, digital, wireless cameras to be used in a stereo configuration with no hardware synchronization necessary between the two cameras. Additionally, the modularity of the system and its compatibility with the RAVEN indoor testbed allows the system to be a platform for rapidly testing full SLAM implementations using the synchronized stereo vision.

5.1 Result Summary

The output of the various modules of this thesis include relative location estimates for features captured by the cameras, detected objects meeting either specific or generic criteria, and an accessible map that contains the features and detected objects. This thesis designs and implements a method for synchronizing stereo cameras using feature trails for corresponding features in several sequential frames from the left and right cameras. The synchronization proves to be effective, reducing error for estimated locations by increasing the accuracy of disparity calculations. The error reduction leads to a significant improvement of the synchronized stereo environment map over the ideal single camera map, as well as the unsynchronized stereo map. For object detection, this thesis provides automatic detection of undefined object

locations with a high success rate. Also provided is a tool to reduce the creation time of object detection classifiers cascades, which detect specific objects. Once created, these classifiers are used by the vision system to detect the objects in real time. Results indicate that the classifiers also have a high success rate at detecting objects matching the pose that they were trained with. This implies that a mission planner should train classifiers for multiple object poses if the planner wants the vision system to be able to detect them.

5.2 Limitations and Future Work

There are many improvements that can be made to each component of this system, mainly arising from the limitations of the implementation using preexisting libraries such as OpenCV for vision, numerical python for feature location estimation, and VPython for visualization of the map.

First, the feature detection used in this thesis is OpenCV's `cvGoodFeaturesToTrack`, which finds features based on corners detected in the image. For more robust detection and for the ability to rerecognize features without landmark localization first, a scale-invariant feature detector, such as SIFT, should be used. Because of the heavy computational requirements of SIFT and the kd-tree or RANSAC algorithms used to match SIFT features, a real-time replacement of the OpenCV feature detector should take advantage of parallel or GPU processing techniques.

Another area for improvement is the synchronization algorithm developed for this thesis. It was sufficient for delays up to half the width of the $N = 5$ frame window, approximately 0.17 seconds. Delays over 0.17 seconds may cause significant estimation errors as the curve fit to the left feature flow is based on points sampled from a curve section that may be different from the samples by the right feature flow. Additionally, the range of motion is restricted to primarily yaws, pitches, and translations over the 0.3 seconds of the 5-frame window. These motions must be dominant over camera roll, because a roll transformation cannot be approximated by a simple translation from the left camera to the right. In practice, moving tests

with the stereo system did not contain roll as the dominant motion. These tests attempted to simulate the behavior of real ground and aerial vehicles, where rolls are typically accompanied by lateral movements when they appear. Unfortunately, there are cases where stereo cameras are not mounted facing the front of a vehicle but facing down or to the side. For these cases, future implementations of the synchronization algorithm could first use visual odometry to determine a roll and apply a correcting transformation. Calculated visual odometry can also be matched with or used in lieu of accelerometer data for full SLAM implementations. Should future projects using stereo synchronization require more than a corresponding set of feature points, the visual odometry-assisted flow estimation can be used to transform the right camera image itself to correspond with the left's. In this way, the synchronized, transformed right camera image could be used for complete stereo depth map calculation or for processing with classifiers.

Finally, the k-means algorithm used for clustering object detection can be improved upon in future implementations by clustering for multiple k and selecting the result n such that $n + 1$ does not produce a significant decrease in average cluster covariance. This will decrease the number of clusters created, especially clusters in close proximity with others. It should allow objects to be identified as single objects rather than as multiple objects with close spatial locality.

Bibliography

[1] *CGI Programming on the World Wide Web.* O'Reilly, 1996.

[2] S. Ahrens. Vision-Based Guidance and Control of a Hovering Vehicle in Unknown Environments. Master's thesis, Massachusetts Institute of Technology, 2008.

[3] B. Bethke. Persistent vision-based search and track using multiple UAVs. Master's thesis, Massachusetts Institute of Technology, 2007.

[4] J.Y. Bouguet. Pyramidal Implementation of the Lucas Kanade Feature Tracker Description of the algorithm. *Intel Corporation, Microprocessor Research Labs*, 2000.

[5] Bouguet, Jean-Yves. Camera Calibration Toolbox for Matlab. Available at `http://www.vision.caltech.edu/bouguetj/calib_doc/index.html`, 2008.

[6] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition.* The MIT Press, September 2001.

[7] DARPA. Darpa Urban Challenge. Available at `http://www.darpa.mil/GRANDCHALLENGE/`, 2008.

[8] Draganfly Innovations Inc. 2.4GHz Micro Wireless Camera Systems and Accessories. Available at `http://www.rctoys.com/rc-products-catalog/RC-PARTS-WIRELESS-VIDEO-EQUIPMENT.html`, 2008.

[9] N. Dussault and E. Frigeri. iRobot introduces Roomba intelligent floorvac-the first automatic floor cleaner in the US Press release, 2002.

[10] A. Eliazar and R. Parr. DP-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks. *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1135–1142, 2003.

[11] Foster-Miller. TALON Military Robots, EOD, SWORDS, and Hazmat Robots. Available at `http://www.foster-miller.com/lemming.htm`, 2008.

[12] General Atomics Aeronautical Systems. Predator Unmanned Aerial Vehicle. Available at `http://www.ga-asi.com/products/predator.php`, 2008.

[13] General Atomics Robotic Systems. Mobile Detection Assessment and Response System (MDARS). Available at `http://www.gdrs.com/robotics/programs/program.asp?UniqueID=27`, 2008.

[14] W.E.L. Grimson. *Object recognition by computer: the role of geometric constraints.* MIT Press Cambridge, MA, USA, 1991.

[15] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007.

[16] D. Hahnel, W. Burgard, D. Fox, and S. Thrun. An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*, 1, 2003.

[17] C. Harris and M. Stephens. A combined corner and edge detector. *Alvey Vision Conference*, 15:50, 1988.

[18] L.J. Heyer, S. Kruglyak, and S. Yooseph. Exploring Expression Data: Identification and Analysis of Coexpressed Genes. *Genome Research*, 9(11):1106, 1999.

[19] Intel Corporation. OpenCV Computer Vision Library. Available at `http://www.intel.com/technology/computing/opencv/`, 2008.

[20] T. Lemaire, C. Berger, I.K. Jung, and S. Lacroix. Vision-Based SLAM: Stereo and Monocular Approaches. *International Journal of Computer Vision*, 74(3):343–364, 2007.

[21] R. Lienhart and J. Maydt. An extended set of Haar-like features for rapid object detection. *Image Processing. 2002. Proceedings. 2002 International Conference on*, 1, 2002.

[22] J.S. Lim. *Two-dimensional signal and image processing.* 1990.

[23] D.G. Lowe. Object recognition from local scale-invariant features. *International Conference on Computer Vision*, 2:1150–1157, 1999.

[24] D.G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[25] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Edmonton, Canada, 2002. AAAI.

[26] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003. IJCAI.

[27] Northrop Grumman Corporation. Global Hawk Unmanned Aerial Vehicle. Available at `http://www.northropgrumman.com/unmanned/`, 2008.

[28] Numerical Python. Available at `http://numpy.scipy.org/`, 2008.

[29] K. Ohno, T. Tsubouchi, and S. Yuta. Outdoor map building based on odometry and RTK-GPS positioning fusion. *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, 1.

[30] Olson, E., Moore, D., and Huang, A. LCM: Lightweight Communications and Marshalling. Available at http://code.google.com/p/lcm/, 2008.

[31] Panasonic Corporation. BL-C131A. Available at `http://www2.panasonic.com/consumer-electronics/shop/Cameras-Camcorders/Network-Cameras/PetCams/`, 2008.

[32] M. Pfingsthorn, B. Slamet, and A. Visser. A Scalable Hybrid Multi-Robot SLAM Method for Highly Detailed Maps. *Proceedings of the 11th RoboCup International Symposium, July*, 2007.

[33] Point Grey Research Inc. Stereo Vision Products. Available at `http://www.ptgrey.com/products/stereo.asp`, 2008.

[34] Python Programming Language. Available at `http://www.python.org/`, 2008.

[35] Python Psyco Module. Available at `http://psyco.sourceforge.net/`, 2008.

[36] Robomow - Automatic Lawn Mowers. Friendly Robotics. Available at `http://www.friendlyrobotics.com/`, 2008.

[37] J.M. Saez and F. Escolano. Entropy minimization slam using stereo vision. *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 36–43, 18-22 April 2005.

[38] J. Shi and C. Tomasi. Good features to track. *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600, 1994.

[39] SICK. SICK Sensor Intelligence. Available at `http://sick.com/home/en.html`, 2008.

[40] S. Sinha, J.M. Frahm, M. Pollefeys, et al. GPU-based Video Feature Tracking and Matching. *EDGE 2006, workshop on Edge Computing Using New Commodity Architectures*, 2006.

[41] Vicon Company. Vicon Motion Capture Systems. Available at `http://www.vicon.com/`, 2008.

[42] Videre Design. Stereo Cameras from Videre Design. Available at `http://www.videredesign.com/vision/stereo_products.htm`, 2008.

[43] Visual Python 3D Programming Module. Available at `http://www.vpython.org/`, 2008.

[44] B. Yamauchi, P. Pook, and A. Gruber. Bloodhound: A Semi-Autonomous Battlefield Medical Robot. *Proceedings of the 23rd Army Science Conference*, 2002.

Appendix A

Software Implementation Details

Besides the camera selection and mounting, the majority of this thesis lies in an implementation of the computer vision system that features synchronizing stereo cameras for the purposes of mapping. While it was designed and intended for ACL's RAVEN development platform, it can be easily exported for use in other environments. This appendix provides abreviated implementations of the non-obvious software in this thesis.

The software design for this system focuses on parallel processing for improved performance and modularity for improved extensibility. The system is written primarily in Python [34], which was chosen for its high-level programming abstractions that lend it for rapid prototyping and modification. In addition, Python possess a large number of consistently maintained libraries with a wide variety of functionality, including MATLAB-like matrix manipulation, image processing, and GUI creation. In terms of performance, the majority of the image processing for this project is handled by the Python bindings to the native OpenCV libraries and the majority of the complex math is also handled by wrappers to native libraries. Since the major bottlenecks of the system are machine vision and matrix manipulation, these native libraries allow the system to run nearly as fast as a compiled language solution. To speed up the interpreted parts of the system, the Python Psyco optimization module [35] was used.

A.1 Image Pull

The image puller is a component that grabs arbitrary images from any Panasonic network camera, thought it is specifically tested with the BL-C131 and BL-C20 cameras. It depends on Python packages `urllib`, `threading`, and `os` if run by itself. The class will operate with either execute a function callback upon image retrieval, which is used for this thesis, or it will create a file to indicate that it has successfully finished downloading an image, which is used as the image downloading component by [2]. File creation is a useful interface for programs written without Python remote procedure call abilities. The message passing is performed instead by the file system. For both of these projects, the image was stored in a filesystem in system memory for speed. If a normal filesystem is used, the callback version of this code should probably use Python module `StringIO`'s in-memory files. An abrieviated form of the code to download images is in Figure A-1. Parameters used by the function include `self.dest`, which should be initialized to the destination for images, `self.callback`, which is a reference to a Python function that is called with the image location when an image is successfully retrieved, and `self.tmpfile`, which is the filename to be used for passing a message via the filesystem.

A.2 Stereo Synchronization

The stereo synchronization for this system is provided by an implementation of the pseudocode in Sec. 2-9. The abbreviated Python implementation of the code using a third order polynomial for the curves is in Figure A-2. This code takes as input right and left time tracks for a single feature in the variables `rxs` and `rys`, respectively. It is dependent on the `numpy`, Numerical Python, module [28]. The code also makes use of precalculated coefficients to populate the gradient and hessian matrices. This is done for efficiency but limits the implementation to third order polynomials.

```
1     urlopener = loginurlopener()
2     urlopener.setpasswd(username,password)
3     fp=urlopener.open(self.source)
4     while(1):
5           #taking advantage of length
6           line=fp.readline()
7           if (line[0:15] == 'Content-length:'):
8                 length = int(line[16:])
9                 # skip Content-type
10                fp.readline()
11                fp.readline()
12                # read the image
13                image = fp.read(length)
14                # save temporary file
15                f=open(self.dest+".tmp","w")
16                f.write(image)
17                f.close()
18                # atomically save the file at the real destination
19                os.rename(self.dest+".tmp", self.dest)
20                self.callback(self.dest)
21                f2 = open(self.tmpfile, "w")
22                f2.close()
```

Figure A-1: Abbreviated Python code to download images from Panasonic network cameras.

A.3 Location Estimator

Figure A-3 is the location estimator that takes as input any number of measurement and global translation vectors. It persists the necessary vectors for an estimate until `reset` is called and only calculates an estimate on a call of the `estimate` function. The lists `x`, `d`, and `weight` keep track of every measurement for possible future use. The class is an implementation of Eq.2.8 from [3]. This estimator assumes that the measurement has already been rotated in the global frame, which is accomplished by the function in Figure A-4, which uses the axis-angle rotation function from the `visual` module, VPython [43]. The function takes as input `meas`, a measurement in form of $\mathbf{r} = \left[p_x - \frac{\text{X}_{\text{RES}}}{2},\ p_y - \frac{\text{Y}_{\text{RES}}}{2}, f\right]$ from Sec. 2.4, `location`, a location with `location.att` attitude and `location.pos` position, and `offset`, a known camera

```
1       rxs=numpy.array(rxs) # turn arrays of feature tracks into numpy arrays
2       rys=numpy.array(rys)
3       fit=numpy.polyfit(lxs,lys,3) # 3rd order polynomial
4       a,b,c,d=fit
5       error=1000
6       lasterror = error
7       iterations = 0
8       vxo,yo=rxs[0],rys[0]
9       lastxo,lastyo=xo,yo
        # Newton's method
10      while (iterations < 10):
11              iterations+=1
12              lasterror = error
13              error = numpy.sum((rys - yo - numpy.polyval(fit,rxs-xo))**2)
14              if (abs(lasterror) < 1.5*abs(error)):
15                      xo=lastxo
16                      yo=lastyo
17                      break
18              dexo = 0
19              deyo = 0
20              st = numpy.polyval([3*a,2*b,c],rxs-xo) # second term
                #note both derivs and hessians have been scaled by 1/2
                #hessian matrix
21              h11=0
22              h11st = numpy.polyval([-6*a, -2*b],rxs-xo) # second term
23              h12=numpy.sum(-st)
24              h21=h12
25              h22=len(rys)
26              for k in range(len(rys)):
27              evec = numpy.polyval([-a,-b,-c,-d-yo+rys[k]],rxs[k]-xo)
27                      dexo += evec*st[k]
28                      deyo += numpy.polyval([a,b,c,d+yo-rys[k]],rxs[k]-xo)
29                      h11 += st[k]**2+evec*h11st[k]
30              lastxo=xo
31              lastyo=yo
32              h11 = numpy.sum(h11)
33              Hinv=numpy.linalg.inv([[h11,h12],[h21,h22]])
34              grad=numpy.array(numpy.vstack((dexo,deyo)))
35              result = numpy.dot(Hinv,grad)
36              xo=xo-numpy.sum(result[0])
37              yo=yo-numpy.sum(result[1])
```

Figure A-2: Python code implementation of camera synchronization for a single point's left and right tracks

offset determined by calibration in Sec. 2.3.1. The function returns a tuple with the rotated measurement, and the offset location. This is useful when using an absolute robot location measured from some point in the robot and cameras that need to be offset relative to this point. Note that the camera offset does not take into account the camera rotation mentioned from 2.3.1 as this step is done by software as soon as the features are determined.

A.4 Data Archive and Replay System

Finally, the software contains functionality to dump data for a test to an archive and then replay the archive at a later time. This allows a single data capture run to be tested with any number of approaches to the stereo synchronization, object detection, or mapping problems. To ensure consistency of runs, a frame received through A.1 triggers a callback to signal the receipt of an image, which is saved in an archive along with a log file detailing the camera it came from, the timestamp, and the corresponding location at that instant. To replay the archive, the log file is read by the replayer, which makes the callbacks into the system and makes the images and corresponding location data available. The replayer will use the `os` module to sleep between calls so that it maintains the relative delay between frames on replay. An abbreviated version of the archiver and replayer is in Figure A-5 and Figure A-6, respectively. The archiver is a method of the main stereo vision class and takes as arguments an `image` to be dumped, the corresponding `camera` label, an associated `location`, a place `dumpdir` to save the archive, and a `logname` to make the logfile. This method depends on the Python `pickle`, `time`, and `opencv.highgui` modules. The replayer is a callback generator and is initalized by the stereo vision class in lieu of A.1 when the system is set to replay mode. The replayer constructor requires a place `dumpdir` to load the archive, and a `logname` to restore the logfile, corresponding to what they are saved as by the archiver. Additionally, the replayer needs a camera callback and location callback, which it uses to replay a previously saved run.

```
1       class minimizer():
2               def __init__(self, callback=None):
3                       self.callback = callback
4                       self.reset()
5               def reset(self):
6                       self.A = numpy.mat(numpy.zeros((3,3)))
7                       self.b = numpy.mat(numpy.zeros((3,1)))
8                       self.lestimate= None
9                       self.x = []
10                      self.d = []
11                      self.weight = []
12                      self.num = 0
13              def getmeasurementcount(self):
14                      return self.num
15              def addmeasurement(self, measurement, location, weight=1):
                        ''' measurement should be a 3-pule, as should location
                        measurement - a vector towards the object
                        (normalized to unit vector in this program)
                        location - global location where measurement taken
                        '''
16                      m=numpy.array(measurement)
                        # normalize m
17                      m=m/linalg.norm(m)
18                      l=numpy.array(location)
                        # add to our arrays of locations
19                      self.d.append(m)
20                      self.x.append(numpy.mat(l).transpose())
21                      self.weight.append(weight)
22                      trix = weight*(self.I-numpy.outer(self.d[-1],self.d[-1]))
23                      self.A += trix
24                      self.b += trix*self.x[-1]
25                      self.num +=1
26              def estimate(self):
                        ''' spits out a global position measurement tuple'''
27                      q=linalg.solve(self.A,self.b)
                        # put them in vicon coordinates where y = front of plane
28                      self.lestimate=copy.copy(q)
29                      return q
30              def lastestimate(self):
31                      return copy.copy(self.lestimate)
```

Figure A-3: Abbreviated Python class that is used to maintain an ongoing estimate of a feature's location based on many measurements.

```
1     def rotateMeasurement(self,meas,location,offset):
2           if (not location):  # default if location not provided
3                 return meas,offset
            #quaternions
4           qw=min(.999,location.att.qw) # prevents division by 0
5           qx=location.att.qx
6           qy=location.att.qy
7           qz=location.att.qz
8           mvec = visual.vector(meas)
9           ovec = visual.vector(offset)
            # Calculate the rotation in axis-angle format
10          rangle = 2*visual.acos(qw)
11          rx = qx / visual.sqrt(1-qw*qw)
12          ry = qy / visual.sqrt(1-qw*qw)
13          rz = qz / visual.sqrt(1-qw*qw)
            # Rebuild vectors
14          raxis = visual.vector(rx,ry,rz)
15          ovec = ovec.rotate(angle=rangle,axis=raxis)
16          mvec = mvec.rotate(angle=rangle,axis=raxis)
17          loc = [location.pos.x+ovec[0],
                   location.pos.y+ovec[1],
                   location.pos.z+ovec[2]]
18          return mvec, loc
```

Figure A-4: Abbreviated Python code to transform a measurement by a given quaternion.

```
1     def dump_image(self, image, camera, location, dumpdir, logname):
      # dump image/location data to directory if we're recording
2     currtime = time.time()
3     filename = "%.3f"% currtime+'-'+camera+".jpg"
4     locationname = "%.3f"% currtime+"-loc.dmp"
5     locationfile = open(dumpdir+locationname, "w")
6     pickle.dump(location, locationfile)
7     locationfile.close()
8     highgui.cvSaveImage(dumpdir+filename, image)
      # save the logs of the location, image locations
9     logfile = open(dumpdir+logname,"a")
10    logentry =
11    logentry['loc'] = locationname
12    logentry['time'] = currtime
13    logentry['img'] = filename
14    logentry['cam'] = camera
15    logfile.write(str(logentry)+"\n")
17    logfile.close()
```

Figure A-5: Abbreviated Python code to save an archive on a callback.

```
class Replayer(threading.Thread):
2       def __init__(self,dumpdir,cb_cams,cb_loc,logname="log"):
3               self.dumpdir = dumpdir
4               self.cb_cams = cb_cams
5               self.cb_loc = cb_loc
6               self.logname = logname
7               self.realtime = realtime
8               threading.Thread.__init__(self)
9       def run(self):
10              self.logfile = open(self.dumpdir+"/"+self.logname)
11              ltime = 0
12              rltime = 0
13              for entry in self.logfile:
                        # load the log entry
14                      logentry = eval(entry)
                        # load the object
15                      dmpfl = open(self.dumpdir+"/"+logentry['loc'])
16                      loc = pickle.load(dmpfl)
17                      dmpfl.close()
                        # send the location data
18                      self.cb_loc(loc)
                        # send camera data
19                      self.cb_cams[logentry['cam']](self.dumpdir+"/"+logentry['img'])
20                      if (ltime and rltime):
21                      timetosleep=(logentry['time']-ltime)-(time.time()-rltime)
                                # if the time difference is significant, sleep to sync
22                              time.sleep(timetosleep-.01) if (timetosleep > .01):
23                      ltime = logentry['time']
24                      rltime = time.time()
25              self.logfile.close()
```

Figure A-6: Abbreviated Python code to replay an archive by making callbacks.

Appendix B

Digital Image Processing for Analog Images

One of the problems encountered has been video quality for the cameras on board vehicles. The received video contains video abnormalities from low camera quality as well as interference in the communication channel. To improve the video to a point where classifiers could reasonably be used to identify objects in the image, an adaptive video processing filter was designed to improve video quality.[1]

B.1 Degradation characteristics

Before designing a noise reduction filter, the noise was characterized that prevents the haar classifier from working correctly. By visual inspection of the raw camera feed and the detection output from the classifier, significant amounts of row-dependent signal noise was discovered to be the main barrier for object detection. This noise was likely introduced by either the camera system, the RF communication channel, or the receiver. When this noise passed over an object in question, it caused the object to take on completely different visual characteristics. An example of an ideal picture for detection and one obscured by the aforementioned noise is in Figure B-1.

The noise tended to be intermittent, often only obscuring part of the image for one

[1]The noise-reduction algorithms in this section were co-designed with Michael Scharfstein.

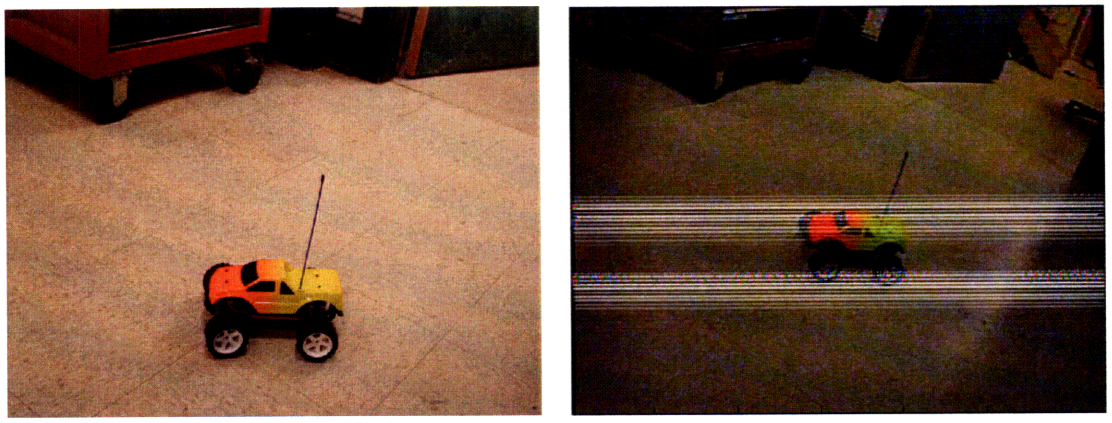


Figure B-1: Difference between an ideal image (left) and the actual video received from the RF video camera.

frame. Additionally, it only affected a fraction of the horizontal lines in a localized region of the image, with relatively unaffected lines coming between affected lines and completely unaffected lines outside the "distorted" regions. The regions contains large high-frequency oscillations of the row mean. A typical example of the unique characteristics of the noise relative to actual picture values can be seen in Figure B-2.

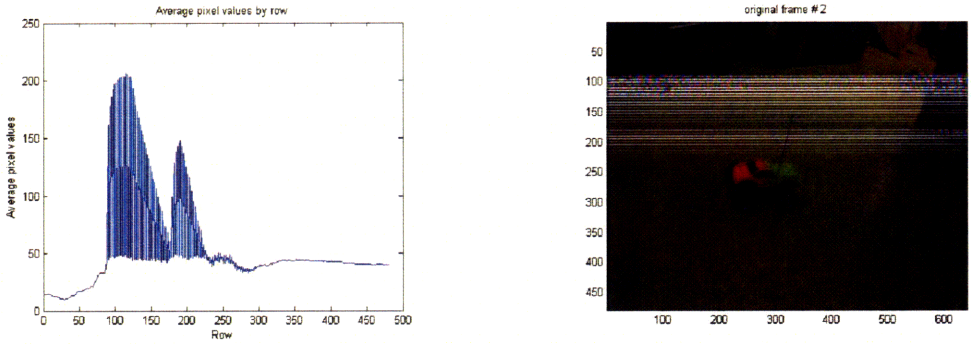


Figure B-2: Average pixel values across each row. Note the average row values that correspond with the unwanted artifacts.

B.2 Noise Reduction Implementation

The approach for restoring the image so that a classifier would detect objects was to detect the distorted regions and to apply an image processing algorithm. Two approaches were used to find the noise and three methods were used in an attempt

to eliminate the noise. The noise reduction algorithms included temporal median filtering, spatial interpolation using undistorted lines, and an adaptive filter that uses a combination of median filtering and spatial interpolation depending on movement in the frame. One limitation placed on this system is performance. Because the image processing used to restore classifier functionality must run in a real-time system, it is limited in complexity and must itself run in real time.

B.2.1 Noise detection

To find the noise, distinct regions with high frequency row-mean oscillations were processed using a matched filter (disturbance detection) and acceptably undistorted lines were isolated within the region by comparing the lines to the corresponding lines in adjacent frames (good line detection). For disturbance detection, the following value was calculated:

$$\frac{d\overline{r}}{dr}\cos\omega r > \text{threshold} \tag{B.1}$$

This value was computed and compared against a threshold. Those values that exceeded the threshold were considered part of the disturbance. Here, ω roughly corresponds to a π, which is roughly one cycle every two lines, and r corresponds to a row in the image.[2]

For good line detection, the following good-line metric was computed:

$$\overline{r_n} - \frac{1}{2}\left(\overline{r_{n-1}} + \overline{r_{n+1}}\right) < \text{threshold} \tag{B.2}$$

where n is the frame number being processed. This was only done within disturbance region found using (B.1). The threshold here was determined by a percentage of the maximum difference in luminosity among lines in the disturbance.

[2]In theory a "real" pattern of alternating lines exactly horizontal to the camera would cause us to mistake the background for noise. This is highly unlikely because very seldom are large numbers of alternating lines exactly perpendicular to the camera in any real environment.

B.2.2 Temporal median filtering

Median filters are excellent at preserving real edges while eliminating random, single-impulse noise. As noise in the video tended to be in one region for a single frame, a median filter in the time domain was selected as a possible candidate for filtering out the noise. The implementation used regions that pass noise detection from (B.1) and (B.2). These regions were median-filtered with the corresponding regions from the previous frame and the next frame:[3]

$$y[x, y, n] = \text{median}(x[x, y, n-1], x[x, y, n], x[x, y, n+1]) \tag{B.3}$$

The limited size and duration of the median filter allowed this approach to image processing to be relatively inexpensive.

B.2.3 Spatial interpolation

Median filters in the time domain are not optimal in all situations. Motion in the frame can cause artifacts in a median-filtered image, as evidenced in the median-filtered image for frame 37 in Figure B-4. While many regions of noise were large, good lines within the disturbance provided data points for an interpolation approach to image restoration. By using the noise-free lines in the image,[4] distorted pixels in the image were interpolated using linear interpolation:[5]

$$y[x, y] = \frac{a}{a+b} x[x, y-a] + \frac{b}{a+b} x[x, y+b] \tag{B.4}$$

Here, a and b are the number of rows to the nearest good line above and below y.

[3] Only 3 frames were used in median filtering to limit the distortion encountered from moving objects.

[4] i.e. lines outside the disturbance region or lines chosen by the good line detector.

[5] Linear interpolation makes sense for computational efficiency and the relatively small scale. More advanced interpolation algorithms were avoided because the real world does not necessarily follow simple contours.

B.2.4 Adaptive filtering

The median filter failed to adequately restore the image when confronted with motion and spatial interpolation failed when faced with a colored version of the expected noise. The colored version of the noise had few acceptable lines to interpolate among with nearly all recognizable data lost. An example of this failure can be found near the top of the bottom-left cell in Figure B-4. From an object detection standpoint, these regions would be better served by a median filter. An adaptive filter was created to decide between the spatial interpolation or the median filter, depending on the "recoverability" of the data through interpolation, a statistic derived from the maximum number of sequential non-good lines and the amount of pixel change between subsequent frames.

B.3 Noise Reduction Results

The disturbance and good line detection algorithms worked remarkably well because of the unique characteristics of the noise. The distinct differences between the noise and legitimate parts of the image not only allowed detection of the noise with a high success rate, but enabled the selective application of image restoration techniques to only affected areas of the image, preserving the details and fidelity of unaffected regions.[6] An example of the disturbance region detection and good line detection can be found in Figure B-3. The results of the two main filtering schemes can be seen in Figure B-4. As in Implementation, various deficiencies in either algorithm was largely complemented by the other through the use of an adaptive filter that selects between the two.

[6]This is rather important. It is counterproductive for algorithms designed to restore classifier functionality to impair it by degrading image quality elsewhere.

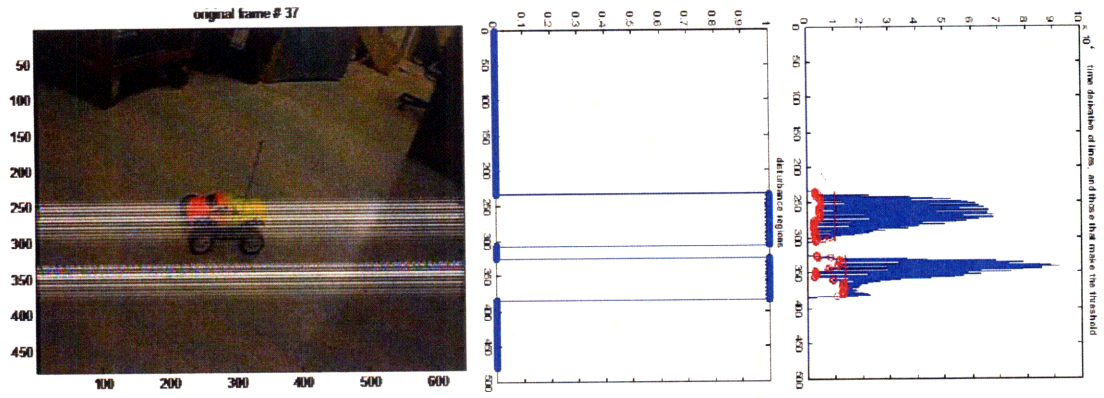


Figure B-3: An example of the noise detection using a matched filter (B.1) and selecting good lines based on differences between adjacent frames (B.2). The time frame differences graphic also shows the threshold levels to decide between "good" and "bad" lines in the image.

B.4 Noise Reduction Conclusions

Two techniques for removing channel interference noise from RF-transmitted digital video were demonstrated. Although the noise was severe in terms of signal-to-noise ratio, most of the noise preventing operation of the object detection classifier could be removed by exploiting the short duration of the noise and its interlaced nature. A median filter in the time domain removed noise remarkably well where there was little movement, but visibly distorted motion in the presence of motion. The intra-frame interpolation method worked well in areas of high motion, but failed to produce a usable image when there was a large area that needed to be interpolated. To use the best features of each filter, an adaptive filter that uses localized motion to choose between the two methods for individual regions of the image was created.

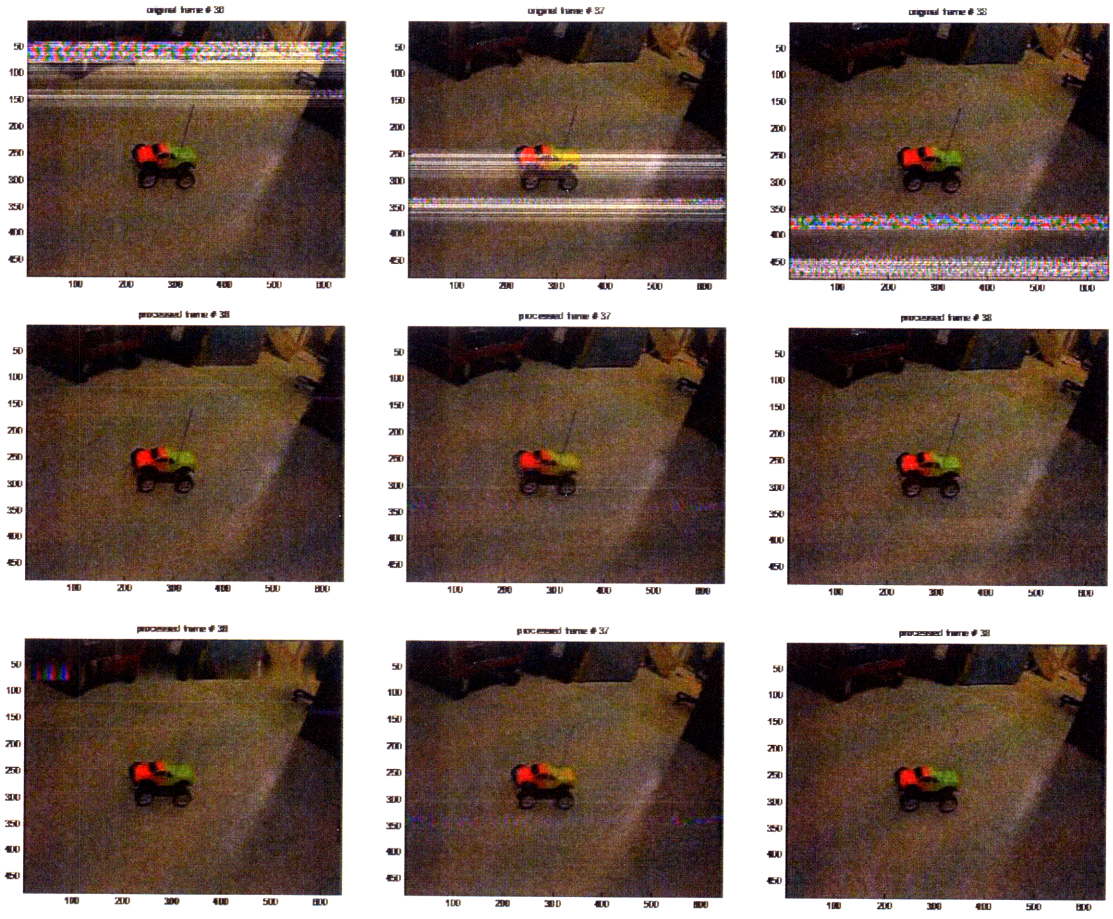


Figure B-4: Before (top) and after for three typical images selectively processed by a temporal median filter (middle) and by spatial interpolation (bottom).